LREC-COLING 2024

Proceedings of the Second Workshop on Computation and Written Language (CAWL 2024) @LREC-COLING-2024

Workshop Proceedings

Editors
Kyle Gorman

May 21, 2024
Torino, Italia

Proceedings of Second Workshop on Computation and Written Language (CAWL 2024) @LREC-COLING-2024

ISBN 978-2-493814-28-9
ISSN 2951-2093 (COLING); 2522-2686 (LREC)

Jointly organized by the ELRA Language Resources Association
and the International Committee on Computational Linguistics

Message from the Chairs

Welcome to the second meeting of the CAWL workshop, featuring eight original paper presentations, an invited talk by Nizar Habash, and an invited lecture by Jalal Maleki.

This year's workshop is sponsored by Google; we thank them for their support.

Since our first meeting in Toronto in July 2023, we have formed a special interest group: the ACL Special Interest Group on Writing Systems and Written Language, or SIGWrit for short. This SIG will be responsible for organizing future meetings of CAWL, and may pursue other objectives as decided by the officers and members of the SIG.

The current *pro tempore* officers of SIGWrit are president Richard Sproat, vice president Emily Prud'hommeaux, secretary-treasurer Kyle Gorman, and student member Noah Hermalin. As per ACL policy, we will organize an election for new officers in fall 2024. More information about the SIG, including its constitution, can be found at `https://sigwrit.org/`.

Invited Talks and Lectures

Nizar Habash: On Writing Arabic

The Arabic language, broadly defined, encompasses a diverse collection of varieties that are tied together historically and linguistically, but with a high degree of variations in terms of phonology, morphology, lexicon, and naturally orthography. In this talk we present a condensed summary of the challenges of writing Arabic and the evolution of different orthographic solutions to address them. The accumulation and persistence of different conventions have led to many co-existing orthographies today creating a complex space of challenges for computational modeling. Among the examples we discuss are subtle differences in Standard Arabic spelling across Arab countries, using scripts other than Arabic for writing Arabic dialects, and, most recently, social media experimentation with reverting to ancient orthographic conventions to fight AI censorship algorithms.

Jalal Maleki: Balancing Linguistic Integrity and Practicality: The Design Journey of Dabire, a Romanized Writing System for Persian

Developing a new writing system, like the romanized Dabire for Persian, requires a nuanced balance between adhering to orthographic principles and making pragmatic compromises. At the core of Dabire's design is the principle of linguistic soundness, with phonemicity as a cornerstone, ensuring a direct, systematic encoding of sounds to simplify the learning process and enhance teaching efficacy. The focus on phonemicity, crucial for the script's ease of use and learning, particularly benefits young learners and non-native speakers. However, the design process also involves balancing phonemic and morphophonemic considerations, acknowledging the complex interplay between adjacent morphemes on sound realization. In practice, rigorous maintenance of both phonemicity and morphophonemic consistency is impossible and concessions are sometimes necessary. Other properties of the Dabire writing system that will be discussed are faithfulness, transparency, completeness and orthographic depth. This talk will highlight the considerations and compromises in designing Dabire, revealing the challenges and opportunities of developing a practical, efficient, and linguistically sound writing system for Persian.

Organizing Committee

- Kyle Gorman, Graduate Center, City University of New York and Google, USA
- Emily Prud'hommeaux, Boston College, USA
- Brian Roark, Google, USA
- Richard Sproat, Google DeepMind, Japan

Program Committee

- David Ifeoluwa Adelani
- Manex Agirrezabal
- Sina Ahmadi
- Cecilia Alm
- Mark Aronoff
- Steven Bedrick
- Taylor Berg-Kirkpatrick
- Amalia Gnanadesikan
- Christian Gold
- Alexander Gutkin
- Nizar Habash
- Yannis Haralambous
- Cassandra Jacobs
- Martin Jansche
- Kathryn Kelley
- George Kiraz
- Christo Kirov
- Jordan Kodner
- Anoop Kunchukuttan
- Yang Li
- Constantine Lignos
- Zoey Liu
- Jalal Maleki
- M. Willis Monroe
- Gerald Penn
- Yuval Pinter
- William Poser
- Shruti Rijhwani
- Maria Ryskina
- Anoop Sarkar
- Lane Schwartz
- Djamé Seddah
- Shuming Shi
- Claytone Sikasote
- Fabio Tamburini
- Kumiko Tanaka-Ishii
- Lawrence Wolf-Sonkin
- Martha Yifiru Tachbelie

Table of Contents

Tutorial Program

Tuesday, May 21, 2024

09:00–09:10	*Opening remarks* Organizers
09:10–10:10	*Invited talk: On Writing Arabic* Nizar Habash
10:10–10:30	*ParsText: A Digraphic Corpus for Tajik-Farsi Transliteration* Rayyan Merchant and Kevin Tang
10:30–11:00	***Coffee break***
11:30–12:00	*A Joint Approach for Automatic Analysis of Reading and Writing Errors* Wieke Harmsen, Catia Cucchiarini, Roeland van Hout and Helmer Strik
12:00–12:20	*Tool for Constructing a Large-Scale Corpus of Code Comments and Other Source Code Annotations* Luna Peck and Susan Brown
12:20–14:00	***Lunch break***
14:00–14:30	*Tokenization via Language Modeling: the Role of Preceding Text* Rastislav Hronsky and Emmanuel Keuleers
14:30–14:50	*Abbreviation Across the World's Languages and Scripts* Kyle Gorman and Brian Roark
14:50–15:20	*Now You See Me, Now You Don't: 'Poverty of the Stimulus' Problems and Arbitrary Correspondences in End-to-End Speech Models* Daan van Esch
15:20–15:40	*Towards Fast Cognate Alignment on Imbalanced Data* Logan Born, M. Willis Monroe, Kathryn Kelley and Anoop Sarkar
15:40–16:00	*Business meeting* Organizers

Tuesday, May 21, 2024 (continued)

16:00–16:30	***Coffee break***
16:30–16:50	*Simplified Chinese Character Distance Based on Ideographic Description Sequences* Yixia Wang and Emmanuel Keuleers

ParsText: A Digraphic Corpus for Tajik-Farsi Transliteration

Rayyan Merchant[*], **Kevin Tang**[†,*]
[*]University of Florida
Department of Linguistics, College of Liberal Arts and Sciences
rayyan.merchant@gmail.com
[†]Heinrich Heine University Düsseldorf
Department of English Language and Linguistics, Faculty of Arts and Humanities
kevin.tang@hhu.de

Abstract

Despite speaking dialects of the same language, Persian speakers from Tajikistan cannot read Persian texts from Iran and Afghanistan. This is due to the fact that Tajik Persian is written in the Tajik-Cyrillic script, while Iranian and Afghan Persian are written in the Perso-Arabic script. As the formal registers of these dialects all maintain high levels of mutual intelligibility with each other, machine transliteration has been proposed as a more practical and appropriate solution than machine translation. Unfortunately, Persian texts written in both scripts are much more common in print in Tajikistan than online. This paper introduces a novel corpus meant to remedy that gap: ParsText. ParsText contains 2,813 Persian sentences written in both Tajik-Cyrillic and Perso-Arabic manually collected from blog pages and news articles online. This paper presents the need for such a corpus, previous and related work, data collection and alignment procedures, corpus statistics, and discusses directions for future work.

Keywords: parallel text, Persian, Tajik, Farsi, orthography, transliteration, Cyrillic, Perso-Arabic

1. Introduction

ParsText is a new digraphic Persian corpus created for the express purpose of transliteration between two Persian dialects and their scripts: Tajik-Cyrillic in Tajikistan and Perso-Arabic in Iran and Afghanistan. The corpus consists of 2,813 sentences, with average Tajik-Cyrillic and Perso-Arabic sentence lengths of 15.00 and 15.57 words, respectively.

To the best of our knowledge, only two previous efforts have investigated machine transliteration between these two scripts thus far, and both of them lacked parallel corpora with which to directly evaluate their models (Davis, 2012; Megerdoomian and Parvaz, 2008). While digraphic texts are available within Tajikistan in print form, similar texts rarely make appearances online, even on the website of Tajikistan's embassy in Iran.[1] ParsText fills this gap as a corpus made up of blog posts and news articles written by native Persian speakers in both scripts. The goal of ParsText is to enable future efforts to train or evaluate their transliteration systems. In an independent study (under review), we use ParsText to train Tajik-Farsi transliteration models. This data will be made available on OSF[2] and Github[3].

In Section 2, the importance of Tajik-Farsi transliteration and why ParsText, a digraphic parallel-text corpus at the sentence level, is preferable to lists with word pairs in isolation is discussed. Section 3 introduces previous and related work. Section 4 describes how the corpus was developed. Section 5 provides corpus statistics and observations. Finally, Section 6 concludes the paper.

2. Background

2.1. Motivation

Tajik Persian (henceforth, Tajik) is the formal register of Modern Persian spoken in Tajikistan. While spoken Tajik has evolved separately for centuries, the formal register retains extremely high levels of mutual intelligibility with the formal Persian of Iran and Afghanistan (both henceforth referred to as Farsi) (Perry, 2005). Unlike these two countries which use the traditional Perso-Arabic script, Tajikistan uses the relatively new Tajik-Cyrillic script due to its Soviet heritage. Proposals have been made to shift Tajik back to the Perso-Arabic script, but any significant shift will likely not occur soon as Tajikistan's former Minister of Culture stated in 2008 that "...some 90-95% of Tajikistan's population is not familiar with Arabic script..." (Ghufronov, 2008). As a result, the vast majority of the 10 million Persian speakers in Tajikistan cannot read written Persian media produced by the 100 million Persian speakers in Iran and Afghanistan. This restriction extends to the Internet, where Farsi dominates. For example, as of September 2023, the Tajik Wikipedia had 269,857 articles and 10.5 million

[1] `https://mfa.tj/tg/tehran`
[2] `https://doi.org/10.17605/OSF.IO/37GZX`
[3] `https://github.com/merchantrayyan/ParsText`

words across all content pages compared to the Farsi Wikipedia's 5.5 million articles and 194 million words (Wikimedia Foundation, 2023b). These two scripts are highly incongruous (Perry, 2005). The Perso-Arabic script, as an impure abjad, often omits vowels, and those that are written are ambiguous. Meanwhile, the Tajik-Cyrillic script, as an alphabet, writes out all vowels, making it a better phonetic representation of the language than the Perso-Arabic script. Table 1 illustrates how the same sentence is represented in both scripts, with a Latin transliteration and an English translation provided for clarity.

Script	**Sentence**
Farsi (Perso-Arabic)	'زبان فارسی'
Tajik (Tajik-Cyrillic)	'забони форсӣ'
Latin Translit.	'zaboni forsī'
English Translation	'The Persian language'

Table 1: Example sentence written in Farsi and Tajik with Latin transliteration and English translation

2.2. Challenges with Typical Tajik-Farsi Parallel Corpora

Although Tajik and Farsi descend from a common root, they have nonetheless diverged in several aspects, including grammar, lexicon, and pronunciation (Perry, 2005). As a result, Tajik and Farsi versions of the same text made in isolation from each other, such as the United Nations Declaration of Human Rights, are often quite divergent (Gacek, 2015). As a result, they do not align on a word-to-word basis and cannot be used for the task of transliteration, the conversion of text in one script to another. Additionally, several discrepancies between Tajik and Farsi mean that any transliteration system must take into account features at the interword level.

The Persian 'Ezafe' is one example of such an interword feature, as a grammatical feature that links a modifier to a preceding head noun (or preceding modifier) (Perry, 2005). In accordance with its phonetic nature, the Tajik standard typically writes the Ezafe as an -и attached to the previous word. In contrast, the Perso-Arabic script often omits the Ezafe, so the reader must infer its location from the surrounding context. Typically, the Ezafe is only written when added to the plural marker ها ('ho'). Otherwise, usually if needed to disambiguate a phrase, it is written as a diacritic at the end of the head noun.

As the Ezafe has the potential to drastically change a sentence's phrasal boundaries, and thus its meaning, detection of the Ezafe is an important step in a Natural Language Processing pipeline for Persian (Asghari et al., 2014; Doostmohammadi et al., 2020).

On a basic level, several affixes are always attached to the stem word in Tajik, but written either separately or conjoined with a Zero Width Non-joiner character (ZWNJ) in Farsi (Megerdoomian and Parvaz, 2008). These discrepancies are what make the transliteration task challenging. Further challenges are described in Appendix A.

3. Related Work

Previous investigations of Tajik-Farsi transliteration systems have made use of non-digraphic datasets, resulting in indirect methods of system evaluation. Megerdoomian and Parvaz (2008) created a Tajik-only dataset from the news site Radio Ozodi, judging performance of Tajik to Farsi transliteration through detection of correctly-spelled Farsi words. Davis (2012) utilized a Tajik-Farsi word list of 3,503 pairs as training data for a statistical transliteration system. To evaluate said system, two unrelated Tajik and Farsi datasets were used. None of these datasets or transliteration systems have been made public.

To the authors' knowledge, only one other dataset has been made publicly available for the same purpose as ParsText: the training data released by Github user *stibiumghost*[4] for a Tajik-to-Farsi transliteration project[5] based on work by Talafha et al. (2021). These data were uploaded to Github on December 2022, well after the ParsText corpus was created in February 2022.

This dataset consists of a 43,535 word dictionary and collection of poetry and news with 404,755 Farsi tokens and 392,562 Tajik tokens. We note that in-depth exploration of this corpus and comparison to our own present avenues for further research. We also believe that our corpus, despite its smaller size, would be appropriate for use as an evaluation dataset in combination with the larger *stibiumghost* dataset.

Beyond Persian, there exist several datasets made for machine transliteration of a single language. For Jordanian Arabic, Talafha et al. (2021) created a dataset in Arabic and a non-standard romanization known as Arabizi. Ahmadi et al. (2022) compiled a corpus of Kurdish news articles written in the Sorani (Arabic-based) and Kurmanji (Latin-based) orthographies. More recently, Gow-Smith et al. (2022) reconstructed part of a 16th-century Scottish Gaelic manuscript in modern orthography. These corpora all focus on low-resource lan-

[4]https://github.com/stibiumghost/tajik-to-persian-transliteration/tree/main/training_data

[5]https://github.com/stibiumghost/tajik-to-persian-transliteration

guages and tackle similar challenges in transliteration due to non-phonetic orthographies.

4. The Corpus

4.1. Data Collection

After an extensive online search, two main sources of parallel data presented themselves: blog pages and British Broadcasting Corporation (BBC) News articles. The two blogs we found were written by native Persian speakers who knew both orthographies and dealt with a wide variety of topics ranging from poetry to politics.[6][7] Latin orthographies for Persian, such as Dabire, were not considered as they are not standard in any Persian-speaking country (Maleki, 2008). These blogs and articles, as opposed to individual word lists, provide inter-word details such as the aforementioned affixes and ezafe which are critical to Tajik-Farsi transliteration. Moreover, they deal with a variety of formal topics, and are therefore written in a formal register of Persian.

To filter out posts that lacked such sentence alignment, as well as those written in only one script or in other languages (usually Russian), we opted to manually collect these data rather than use an automatic website scraping tool.

We were also able to find 23 BBC News articles written in both orthographies[8][9][10] during the time BBC Tajik operated from 1993 to 2015 (BBC, 2015). These articles almost exclusively deal with politics, and exhibited a similar degree of word-to-word alignment. Due to the small number of articles, we decided to collect these manually.

As the first author is a non-native speaker of Persian, he conducted manual inspection of texts for word-to-word alignment during collection, along with spot checking at later points. In this manner, texts that did not meet this standard were filtered out as well. A few sample sentences from ParsText are available in Appendix B.

4.2. Data Processing

As the corpus that we compiled was not aligned on a sentence-to-sentence basis, we aligned each individual source document with GaChalign[11], a Python implementation of the Gale-Church alignment algorithm (Tan and Bond, 2014; Gale and Church, 1993). We note that our corpus presents some inconsistencies due to differences in the authors' word choice, and has not undergone in-depth analysis from native speakers to be corrected. Experimentation with the data uploaded on Github by *stibiumghost* revealed that those present similar inconsistencies. Creation of a digraphic corpus rigorously checked by native Persian speakers therefore presents another avenue for further research.

[6]http://dariussthoughtland.blogspot.com/

[7]http://jaamjam.blogspot.com/

[8]https://www.bbc.com/tajik

[9]https://www.bbc.com/persian/indepth/cluster_tajikistan_page

[10]https://www.bbc.com/persian

[11]https://github.com/alvations/gachalign

5. Statistics and Observations

In the absence of lemmatization tools for Tajik, token - rather than lemma - statistics of the corpus are presented in this paper. Table 2 lists corpus statistics, while Table 3 provides the top ten most frequent tokens in both scripts.

Statistics	**Farsi**	**Tajik**
# of sentences	2,813	2,813
# of word tokens	43,846	42,226
# of characters	186,414	222,986
Avg. # of tokens in a sentence	15.57	15.00
Avg. # of characters in a token	66.15	79.13

Table 2: ParsText Statistics. Note that any character statistic does not include whitespace characters.

In accordance with the fact that Farsi does not have a phonetic orthography, the Farsi character statistics are lower than the Tajik character statistics. However, the token measures are larger, likely reflecting how several Persian affixes and function words are written attached to the preceding word in Tajik, but separately in Farsi.

From Table 3, several observations can be made. First, the top 10 most frequent tokens in Tajik and Farsi are the exact same Persian words. Furthermore, the order of these tokens is also mostly shared with the exceptions of ва / و (English: 'and') and аст / است (English: 'is'). These likely differ in frequency as both words are expressed in multiple ways in both orthographies. For example, аст / است can be attached to the previous word in Tajik but is always written separately in Farsi. Meanwhile, و can be written either separately or attached to the previous word in Farsi. Its two Tajik equivalents, ва and у, are written separately or attached, respectively.

We also note that all but one of the top ten most frequent Tajik and Farsi tokens in ParsText are stop words, with the ninth most frequent token being тоҷикистон/تاجیکستان (English: 'Tajikistan'). While stop words typically do not indicate alignment, in the case of our digraphic, word-to-word corpus, the fact that the Farsi token frequencies are generally very close to their Tajik equivalents

Farsi				Tajik			
Token	Transliteration	Translation	Frequency	Token	Transliteration	Translation	Frequency
و	va, u	‘and’	2,096	дар	dar	‘in’	1,495
در	dar	‘in’	1,498	ба	ba	‘to’	1,323
به	ba	‘to’	1,307	ва	va	‘and’	1,230
که	ki	‘that’ (Conj.)	1,212	ки	ki	‘that’ (Conj.)	1,216
از	az	‘from	1,154	аз	az	‘from’	1,149
این	in	‘this’	883	ин	in	‘this’	910
است	ast	‘is’	636	бо	bo	‘with’	448
با	bo	‘with’	458	аст	ast	‘is’	394
تاجیکستان	tojikiston	‘Tajikistan’	428	тоҷикистон	tojikiston	‘Tajikistan’	393
بود	bud	‘was’	290	буд	bud	‘was’	285

Table 3: Top 10 Most Frequent Farsi and Tajik Tokens in ParsText

indicate that the same wording is being used. As under-resourced languages, few Farsi stop word lists exist, and we know of only one for Tajik. Ensuring removal of the exact same stop words requires a digraphic list of Tajik/Farsi stop words. This task is best left to native speakers.

Additionally, although one would expect the frequency of ’Tajikistan’ to indicate abnormalities, further manual inspection revealed no unnatural occurrences of the word. As such, we believe this is a natural reflection of the provenance of these texts. As the sources all focus on Tajikistan, it appears that the frequency of this proper noun has become similar to that of a pronoun.

To analyze ParsText on a character level, the most frequent trigraphs (character trigrams) were also calculated. To ensure that three-letter tokens did not overpopulate this list, trigraph frequency was conducted over all word types, rather than tokens. The character ’#’ is inserted at word-initial and word-final postion. These data are presented in Tables 4 and 5.

Trigraph	Transliteration	Frequency
а н д	a n d	534
р о #	r o #	519
# т а	# t a	506
н и #	n i #	463
# м а	# m a	410
о и #	o i #	403
# н а	# n a	351
о н и	o n i	319
# м у	# m u	296
и и #	i i #	217

Table 4: Top 10 Most Frequent Tajik Trigraphs in ParsText (Calculated Over Word Types)

Based on the above trigraphs, several more observations can be made. First, the Persian subject marker ’ро’ / ’را’ (’ro’) appears to be well represented, being present as (’r o #’) among the Tajik trigraphs and (’r a #’) and ’r a _’) among the Farsi trigraphs. The two Farsi forms demonstrate that the ZWNJ is not always used by native speakers. The Ezafe also appears to be present in the Tajik list as

Trigraph	Transliteration	Frequency
ا ن #	a n #	528
را#	r a #	528
ر ا _	_ r a	431
ای#	a i #	424
ه ا ی	h a i	420
ه ا _	h a _	392
ن د #	n d #	392
م ی _	m i _	384
# م ی	# m i	356
ی - #	i - #	343

Table 5: Top 10 Most Frequent Farsi Trigraphs in ParsText (Calculated over Word Types)
Note: The Perso-Arabic text must be read right-to-left and the ZWNJ is denoted with ’_’.

(’n i #’, ’o n i’, and ’i i #’) and the Farsi list as (’a i #’ and ’h a i’). Altogther, these trigraph frequencies greatly differ, demonstrating the large contrasts between the Tajik-Cyrillic and Perso-Arabic script. To provide a different picture, we conduct a different set of trigraph frequencies without any vowels, included Appendix C. However, as several Tajik consonants have multiple (up to four) equivalents in Farsi, this does necessarily result in a clearer picture.

6. Conclusion and Future Work

This paper presented ParsText, a corpus of 2,813 digraphic Persian sentences written by native speakers in the Tajik-Cyrillic and Perso-Arabic orthographies. ParsText contains manually-collected data from blog pages and BBC news articles. Based on manual inspection by a non-native speaker and analysis of most frequent tokens, we confirmed ParsText exhibits word-to-word alignment, a crucial requirement for direct evaluation of Tajik-Farsi transliteration systems that is unavailable in other parallel corpora. As such, it enables direct evaluation of Tajik-Farsi machine transliteration efforts. The corpus is available on OSF.[12]

[12]https://doi.org/10.17605/OSF.IO/37GZX

7. Bibliographical References

Sina Ahmadi, Hossein Hassani, and Daban Q. Jaff. 2022. Leveraging Multilingual News Websites for Building a Kurdish Parallel Corpus. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, 21(5).

Habibollah Asghari, Jalal Maleki, and Heshaam Faili. 2014. A probabilistic approach to Persian ezafe recognition. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, volume 2: Short Papers*, pages 138–142, Gothenburg, Sweden. Association for Computational Linguistics.

BBC. 2015. Поёни фаъъолияти сафҳаи сириллики бахши форсии Би-би-сӣ.

Chris Irwin Davis. 2012. Tajik-Farsi Persian transliteration using statistical machine translation. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 3988–3995, Istanbul, Turkey. European Language Resources Association (ELRA).

Ehsan Doostmohammadi, Minoo Nassajian, and Adel Rahimi. 2020. Persian Ezafe Recognition Using Transformers and Its Role in Part-Of-Speech Tagging. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 961–971, Online. Association for Computational Linguistics.

Tomasz Gacek. 2015. Some comments on a parallel text in Dari, Tojiki and Farsi. In Anna Krasnowolska and Renata Rusek-Kowalska, editors, *Studies on the Iranian World*, volume 2, chapter Medieval and Modern, pages 23–34. Jagiellonian University Press, Kraków.

William A. Gale and Kenneth W. Church. 1993. A program for aligning sentences in bilingual corpora. *Computational Linguistics*, 19(1):75–102.

Daler Ghufronov. 2008. Shifting Tajik writing system to Arabic script takes a lot of time, says minister. *ASIA-Plus*.

Edward Gow-Smith, Mark McConville, William Gillies, Jade Scott, and Roibeard Ó Maolalaigh. 2022. Use of Transformer-Based Models for Word-Level Transliteration of the Book of the Dean of Lismore. In *Proceedings of the 4th Celtic Language Technology Workshop within LREC2022*, pages 94–98, Marseille, France. European Language Resources Association.

Jalal Maleki. 2008. A romanized transcription for persian.

Karine Megerdoomian and Dan Parvaz. 2008. Low-density language bootstrapping: the case of tajiki Persian. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA).

John R. Perry. 2005. *A Tajik Persian Reference Grammar*. Brill, Leiden, The Netherlands.

Bashar Talafha, Analle Abuammar, and Mahmoud Al-Ayyoub. 2021. Atar: Attention-based LSTM for Arabizi transliteration. *International Journal of Electrical and Computer Engineering (IJECE)*, 11(3):2327–2334. Number: 3.

Liling Tan and Francis Bond. 2014. NTU-MC Toolkit: Annotating a Linguistically Diverse Corpus. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: System Demonstrations*, pages 86–89, Dublin, Ireland. Dublin City University and Association for Computational Linguistics.

Wikimedia Foundation. 2023a. List of wikipedias.

Wikimedia Foundation. 2023b. Statistics.

A. Challenges in Tajik-Farsi Transliteration

As previously described, Farsi and Tajik diverge in a number of ways which render one-to-one letter conversion largely ineffective. An example transliteration employing such a technique can be seen in Table 6.

Farsi	**Farsi Translit.**	**Tajik**	**Tajik Translit.**
من کتاب را خواندم	mn ktob ro xwondm	ман китобро хондам	man kitobro xondam

Table 6: One-to-one Transliteration of Farsi and Tajik

Owing to the incongruous natures of the two scripts, Perso-Arabic an imperfect abjad and Tajik-Cyrillic an alphabet, many characters map to a single character and vice versa.

A.1. Vowels

The character ا, known as *alef*, can represent several different vowels as demonstrated in Table 7.

The letter و, known as *vav*, can map to the vow-

Farsi	Farsi Translit.	Tajik	Tajik Translit.	English
انجمن	anjmn	анҷуман	anjuman	'organization'
انتخاب	antxob	интихоб	intixob	'choice'
امید	amyd	умед	umed	'hope'
او	aw	ӯ	ü	'(s)he'
آهنگ	ohng	оҳанг	ohang	'song'
خاردن	xoridn	хоридан	xoridan	'to itch'

Table 7: Examples of Alef mapping to various vowels

Farsi	Farsi Translit.	Tajik	Tajik Translit.	English
ولایت	wlayt	вилоят	viloyat	'oblast'
آورد	owrd	овард	ovard	'brought'
گاو	gaw	гов	gov	'cow'
بود	bwd	буд	bud	'was'
امروز	amrwz	имрӯз	imrüz	'today'

Table 8: Examples of Vav mapping to vowels and consonants

els у and ӯ, or the consonant в, as shown in Table 8.

The letter ی, known as *ye*, maps to several different vowels, as seen in Table 9.

Farsi	Farsi Translit.	Tajik	Tajik Translit.	English
یراق	yroq	яроқ	yaroq	'weapon'
دریا	drya	дарё	daryo	'river/sea'
چای	çay	чой	çoy	'tea'
ایران	ayran	Эрон	Eron	'Iran'
خیلی	xyly	хеле	xele	'very'
عالی	'aly	олӣ	olī	'great'
حتی	hty	ҳатто	hatto	'even'

Table 9: Examples of Ye mappings

The consonant ه, known as *he (do cheshm)*, maps to either the consonant ҳ or to vowels when in word final position, as shown in Table 10.

The character ع, or *ayn*, can map to any vowel (see Table 11), and is also inconsistently written.

The ء, or *hamza*, exhibits similar behavior to *ayn* by mapping to both vowels and the Tajik glottal stop sign. It can be written as a standalone letter, or over *alif*, *vav*, or *ye*. However, this character is often replaced with a *ye* or simply removed from the letter it is written over.

Vowel diacritics are often unwritten in the Perso-Arabic script, further obfuscating short vowel determination. Without vowel diacritics, the word گرد may represent either гард /gard/ ('dust'), гирд /gird/ ('round'), or гурд /gurd/ ('hero').

A.1.1. Consonants

As the Perso-Arabic script retains many redundant Arabic consonants from Arabic, some Cyrillic letters each have multiple Perso-Arabic letter equivalents. Table 12 provides an overview of these.

Outside of these redundant consonants and the vowels mentioned previously, consonant to consonant mapping between the two scripts is one-to-one and can be considered trivial.

Farsi	Farsi Translit.	Tajik	Tajik Translit.	English
به	bh	ба	ba	'to'
که	kh	ки	ki	'that (conj.)'
چه	çh	чи	çi	'what'
قاعده	qa'dh	қоида	qoida	'rule'
سیاه	syah	сиёҳ	siyoh	'black'
ده	dh	даҳ	dah	'ten'
فربه	frbh	фарбеҳ	farbeh	'fat'

Table 10: Examples of He Mapping

Farsi	Farsi Translit.	Tajik	Tajik Translit.	English
عضو	'zw	узв	uzw	'limb'
علامت	'lamt	аломат	alomat	'sign'
فعالیت	f'alyt	фаъолият	fa'oliyat	'activity'
ساعت	sa't	соат	soat	'hour'
تاریخ	taryx	таърих/торих	ta'rix/torix	'history'
قرآن	qron	Қуръон	Qur'on	'Quran'

Table 11: Examples of Ayn mapping

B. ParsText Sample Sentences

These example sentences have been lowercased.

(1) вай гуфтааст ки донишгоҳҳои табрез низ омодаи пазириши донишҷӯёни тоҷик ҳастанд

وی گفته است که دانشگاههای تبریز نیز آماده پذیرش دانشجویان تاجیک هستند

(2) як сол пеш аз таваллуди мирзо фатҳъалӣ падараш аз ин мақом барканор шуда буд

یک سال پیش از تولد میرزا فتحعلی پدرش از این مقام برکنار شده بود

(3) мутмаъинам ки мардуми тоҷикистон ҳам аз аҳволи эрон нигарон ҳастанд

مطمئنم که مردم تاجیکستان هم از احوال ایران نگران هستند

C. Trigraphy Frequencies with Vowels Removed

We provide trigraph frequencies excluding all in Tajik and Farsi in this appendix. For Tajik, the letters removed were у, е, ҳ, ъ, а, о, э, я, и, й, ӣ and ю. For Farsi, the letters removed were ا, آ, و, ع, and ی. Tables 13 and 14 show the trigraph frequencies in Tajik and Farsi.

Phoneme	Tajik	Farsi
/z/	з	ز
		ذ
		ض
		ظ
/s/	с	س
		ص
		ث
/t/	т	ت
		ط
/h/	ҳ	ح
		ه

Table 12: One to many Mappings of Consonants from Tajik to Farsi

Trigraph	Transliteration	Frequency
н д #	n d #	329
с т #	s t #	219
т р #	t r #	140
# б р	# b r	134
р н #	r n #	124
т н #	t n #	107
с т н	s t n	94
# ф р	# f r	94
# с р	# c r	94
# м р	# m r	94
д н #	d n #	94

Table 13: Most Frequent Tajik Trigraphs in Pars-Text (Without Vowels)

Trigraph	Transliteration	Frequency
_ ر #	_ r #	395
ن د #	n d #	346
# م _	# m _	204
س ت #	s t #	140
# ب ر	# b r	139
ر ن #	r n #	118
# ن م	# n m	104
# ف ر	# f r	104
ت ن #	t n #	97
ت ر #	t r #	96

Table 14: Most Frequent Farsi Trigraphs in Pars-Text (Without Vowels)
Note: The Perso-Arabic text must be read right-to-left and the ZWNJ is denoted with '_'.

A Joint Approach for Automatic Analysis of Reading and Writing Errors

Wieke Harmsen, Catia Cucchiarini, Roeland van Hout, Helmer Strik
Centre for Language Studies
Radboud University Nijmegen, The Netherlands
{wieke.harmsen, catia.cucchiarini, roeland.vanhout, helmer.strik}@ru.nl

Abstract

Analyzing the errors that children make on their ways to becoming fluent readers and writers can provide invaluable scientific insights into the processes that underlie literacy acquisition. To this end, we present in this paper an extension of an earlier developed spelling error detection and classification algorithm for Dutch, so that reading errors can also be automatically detected. The strength of this algorithm lies in its ability to detect errors at Phoneme-Corresponding Unit (PCU) level, where a PCU is a sequence of letters corresponding to one phoneme. We validated this algorithm and found good agreement between manual and automatic reading error classifications. We also used the algorithm to analyze words written by second graders and words read by first graders. The most frequent PCU errors were *ei*, *eu*, *g*, *ij* and *ch* for writing, and *v*, *ui*, *ng*, *a* and *g* for reading. This study shows how a joint approach for the automatic analysis of reading and writing errors can be implemented. In future research the value of this algorithm could be tested by analyzing corpora containing initial reading and writing data from the same children.

Keywords: Automatic spelling and reading error detection, Reading and writing instruction, Child read speech corpora, Child written language corpora, Phoneme-grapheme alignment, Dutch

1. Introduction

Reading and writing are both skills that children acquire after long periods of intensive instruction and practice. In this sense, reading and writing are essentially different from speaking and listening, which are skills that children spontaneously acquire in daily interaction. The processes of learning to read and write require children to engage in long and sustained practice, preferably under teachers' guidance. During practice, children inevitably make reading and writing errors that teachers need to correct to make children aware of their gaps in knowledge and to help them improve their reading aloud and writing skills. Analyzing the errors that children make on their ways to becoming fluent readers and writers can provide invaluable scientific insights into the processes that underlie literacy acquisition.

Analyzing the child development of reading and spelling errors is not an easy task. There are three important reasons that make this task challenging. Firstly, there is no corpus available that contains longitudinal reading and writing data from the same children. Secondly, a classification scheme that can capture a large variety of both reading and writing errors has not yet been developed. Thirdly, the task of detecting and classifying reading and writing errors manually is laborious and time-consuming.

Recent developments in the field of language and speech technology have made it possible to overcome these challenges partially. For Dutch, a medium-sized language, there are four corpora available that can be used for either reading error or writing error analysis. These are JASMIN, a small corpus of Dutch and Flemish child speech (Cucchiarini et al., 2006), CHOREC, a corpus of Dutch speech by Flemish elementary school children (Cleuren et al., 2008), BasiScript, a corpus of written texts and dictations produced by children in primary school (Tellings et al., 2018a), and DART, a larger corpus of child read speech by first graders (6-7 years old) (Bai et al., 2022). These corpora make it possible to develop scientifically and pedagogically sound error classification schemes, as well as algorithms that apply these schemes to analyze both reading aloud and writing data automatically.

Using a joint classification scheme opens up opportunities for research on literacy acquisition in which reading and writing development are investigated in combination to gain insight into their differences and their interaction. This type of research would profit enormously from longitudinally collected data collections on reading and writing of course, but such databases are not available for Dutch, although we hope that these might be come about in the near future. In any case, an important prerequisite for comparing reading and writing skills is the development of a joint classification scheme that captures both reading aloud and spelling errors. For developing such a scheme, it is not necessary to have reading and writing data of the same children.

Our aim is to present a joint classification scheme for Dutch reading and spelling errors, together with an algorithm that can automatically detect and classify these errors in corpora of read speech and written language. As a first step in this direction, we expanded an existing spelling error detection and classification algorithm (Harmsen et al., 2021b,a) so that it can also detect and classify reading errors. In addition, we applied this expanded algorithm to separate corpora of reading and spelling data from children in Dutch primary school. We describe the type and frequency of reading and spelling errors that we found.

2. Background

2.1. Three essential competences for reading and spelling in Dutch

When children learn to read and write, three competences become relevant. In the first place, phonological awareness (e.g., van Druenen et al. (2019)), which is knowing that words are built from phonemes (i.e., sounds). Phonologically aware learners are able to segment words into phonemes (auditory analysis) and to combine phonemes into words (auditory synthesis).

Written Dutch uses the Latin alphabet, so the second competence involved is knowledge of the alphabetical principle: a grapheme (i.e., single letter) or sequence of graphemes represents a phoneme and vice versa. Borgwaldt et al. (2004) compared the orthographic transparency of five languages that use the Latin alphabet. A transparent orthography is defined as an orthography with both a high feedforward consistency (one-to-one grapheme to phoneme mappings) and high feedback consistency (one-to-one phoneme to grapheme mappings). They conclude that Dutch orthography has as intermediate transparency. In addition, Dutch has a higher feedforward consistency than feedback consistency. This is one reason why reading in Dutch is considered to be less difficult than spelling (Bosman and Van Orden, 1997).

Finally, the child has to acquire an explicit morphological awareness, since Dutch morphological principles are part of the writing system. This may result in the phenomenon that words are pronounced differently than one would expect based on their written form and the set of learned phoneme-grapheme mappings. For example, the verb *hij vindt* (he finds) consists of two morphemes: *vind* and *t*, since it is constructed by taking the root *vind* and adding the third person singular suffix *t*. In this verb, the *dt* is pronounced as a single /t/[1], which means that the pronunciations of *vind* and *vindt* are exactly the same: /v I n t/ (due to final devoicing, the *d* in the root *vind* is pronounced as /t/).

[1]All phonetic transcriptions in this paper are written between slashes and in the computer phonetic alphabet CGN2 (Gillis, 2001).

2.2. Existing classification schemes

So far, each study researching reading or spelling errors in Dutch used its own classification scheme. Most classification schemes manually labeled each misspelled word with a label describing the reading or spelling error (e.g. Kleijnen (1997); Cleuren et al. (2008); Tellings et al. (2018b); Limonard et al. (2020)). A disadvantage of this approach is that it is not clear which part of the word is written incorrectly, and which letters are substituted, deleted or inserted in comparison with the target word.

Another type of classification scheme that was mainly used in research on automatic pronunciation assessment in alphabetic languages, defined reading errors as phonemes that are inserted, deleted or substituted in comparison with the phonetic transcription of the target word (e.g., Zhang et al. (2021); Lin and Wang (2022); Gelin et al. (2023)). These studies were able to return the phoneme that was read incorrectly, but not the letters in the target word that represented this phoneme. That means that important diagnostic information was missing.

2.3. Phoneme-Corresponding Units

In Dutch, a single phoneme can be represented by one or a sequence of two or even more graphemes. In line with Laarmann-Quante (2016), we refer to these grapheme representations as Phoneme-Corresponding Units (PCUs). For example, the Dutch word *maan* (moon) consists of three phonemes /m a n/, and thus three PCUs: *m*, *aa* and *n*. The Dutch word *bureau* (desk) consists of four phonemes /b y r o/, and thus four PCUs: *b*, *u*, *r* and *eau*. Unfortunately the number of phonemes and the PCUs a word consists of are not always equal. Sometimes, a word can have more PCUs than phonemes. This is possible because some graphemes in Dutch are not pronounced. An example is the diminutive name of cupboard *kastje* (*kast* (noun) + *je* (diminutive suffix), little cupboard) with phonetic transcription /k A s j @/ and PCU segmentation *k*, *a*, *s*, *t*, *j*, *e*. In this example, the PCU *t* is not pronounced.

2.4. Primary PCU-phoneme mappings and sound pure words

A distinction can be made between primary and secondary PCU-phoneme mappings. The primary PCU-mappings mark a fixed, unique link between a PCU and a phoneme. They are the PCU-phoneme mappings that are taught initially to beginning readers and writers in first grade of primary school. The primary PCU-phoneme mappings are

VOWELS		CONSONANTS	
Phoneme	PCU	Phoneme	PCU
/a/	aa	/b/	b
/o/	oo	/d/	d
/y/	uu	/f/	f
/e/	ee	/h/	h
/i/	ie	/j/	j
/A/	a	/k/	k
/O/	o	/l/	l
/U/	u	/m/	m
/E/	e	/n/	n
/I/	i	/N/	ng
/EI/	ei	/p/	p
/EI/	ij	/r/	r
/EU/	eu	/s/	s
/UI/	ui	/t/	t
/u/	oe	/v/	v
/AU/	au	/w/	w
/AU/	ou	/x/	g
/@/	e (schwa)	/x/	ch
		/z/	z

Table 1: Dutch primary PCU-phoneme mappings.

presented in Table 1. This table contains for each phoneme just one way to write it, except for the phonemes /EI/ and /OU/, which can be written in two ways. Words containing only primary PCU-phoneme mappings are called *sound pure* words (*klankzuiver* in Dutch).

Not all words can be written correctly using only primary PCU-phoneme mappings. For example, the Dutch language also contains loanwords, words containing a schwa, and words whose spelling depends on morphology. In these cases, secondary PCU-phoneme mappings are used to write these words correctly. Examples of secondary PCU-phoneme mappings are: *e*-/@/ in *bestaan* (to exist) and *eau*-/o/ in *bureau* (desk).

2.5. Dutch PCU-based spelling error detection and classification

In an earlier study (Harmsen et al., 2021a,b), an algorithm was presented that could detect spelling errors by aligning the realized spelling (including spelling errors) with the target spelling using the phonetic transcription of the target spelling. In this algorithm, a spelling error was defined as an inserted, deleted or substituted PCU.

We illustrate the strength of this algorithm with an example. Where a Levenshtein-based alignment algorithm would most likely detect two errors in the misspelling *lag* of the word *lach* (laugh), namely a substitution of *c* with *g* and a deletion of *h*, the newly presented algorithm that also uses the phonetic transcription of the target (i.e., /l A x/) could recognize that both the *ch* and *g* correspond to the same phoneme /x/ and align them with each other. In this way, the detected spelling error is defined as a substitution of the PCU *ch* pronounced as /x/) with *g*, which is more informative.

2.6. The current research

Given that our aim is to develop a joint classification scheme for reading aloud and spelling, we address the following more specific research questions:

1. To what extent can we accommodate and expand an automatic spelling error detection and classification algorithm (Harmsen et al., 2021a,b) in such a way that it is able to detect sound pure reading errors on the basis of phonetic transcriptions of audio recordings?

2. We address three subquestions to make a comparison between reading and spelling:

   (a) Which sound pure PCUs are most frequently written incorrectly by initial writers?

   (b) Which sound pure PCUs are most frequently read incorrectly by initial readers?

   (c) How do these patterns compare?

3. Method

3.1. Data sets

The reading data set consists of phonetic transcriptions of audio that were collected within the project *Dutch Automatic Reading Tutor (DART)* (Bai et al., 2020). In this project, grade 1 pupils (6-7 years old) practiced reading for six weeks (twice a week for 10 minutes) with a system that provided feedback on their reading aloud. Before and after these practice weeks, each pupil had to take three pretests and three posttests. Each test consisted of a list of 24 words that the pupils had to read in one go while their speech was recorded.

For the current study, we selected phonetic transcriptions of 28 audio recordings from the DART pretest and posttest dataset. An annotator made the phonetic transcriptions of the audio. In total, the reading data consisted of phonetic transcriptions of 672 words.

The written data set used in this study consists of 2352 dictations from the BasiScript corpus (Tellings et. al., 2015) that were written by grade 2 pupils (7-8 years old). Each pupil wrote the same dictation, consisting of 25 words. The dictations were originally handwritten by the pupils, digitized (typed) and stored in the BasiScript corpus. In total, the writing data consisted of 58,800 words.

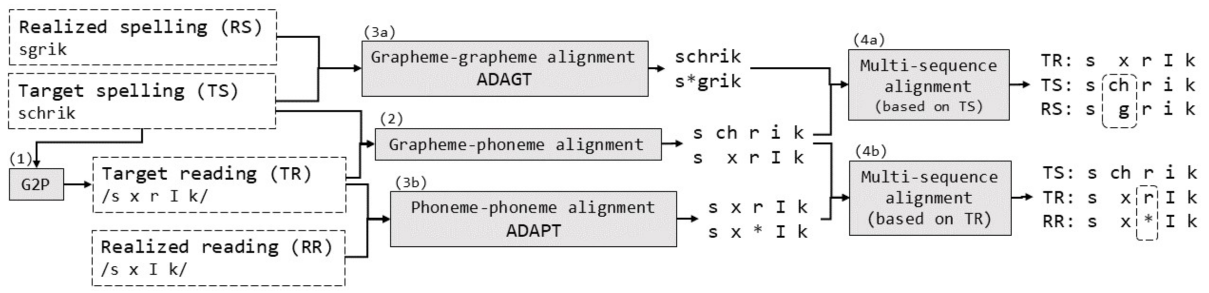

Figure 1: An example of application of the spelling and reading error detection algorithm on the word *schrik* (fright), spelled incorrectly as *sgrik* and read incorrectly as /s x I k/.

3.2. Spelling and reading error detection

To automatically detect reading and spelling errors at PCU-level in the realized readings and spellings, we extended an earlier presented algorithm developed for automatic spelling error detection (Harmsen et al., 2021a). This spelling error detection algorithm consists of four parts. To be able to detect reading errors at PCU-level, we had to create separate variants of parts 3 and 4 of the algorithm. Figure 1 visualizes the four parts in the analysis using an example.

First, the phonetic transcription of the target spelling was obtained automatically using a Dutch grapheme-to-phoneme converter (G2P) webservice (Ten Bosch, 2019). This is the target reading. Secondly, the target reading and target spelling are aligned. This is done using a dictionary that defines all possible ways a phoneme can be written in Dutch. For example, the phoneme /x/ can be written as *g* and as *ch*. Starting from the third step, the analysis procedure is different for the spelling and reading error detection. With respect to reading error detection, the order of the phonemes of the target reading and realized reading is first reversed, and subsequently aligned using the *Algorithm for Dynamic Alignment of Phonetic Transcriptions (ADAPT)* (Elffers et al., 2013). In this algorithm, articulatory features are incorporated to define the distance between two phonemes (Cucchiarini, 1993, 1996). The resulting alignment is again reversed, so that the phonemes are in the correct order again. The readings are reversed, because we want the target reading to match with the final attempt of the speaker to read a word. For spelling error detection, step three consists of aligning the graphemes of the target spelling and realized spelling with each other using the *Algorithm for Dynamic Alignment of Graphemic Transcriptions (ADAGT)* (Bai et al., 2021; Harmsen et al., 2021a). This is an adaptation of ADAPT, made suitable for grapheme alignment. This algorithm aligns vowels only with vowels and consonants only with consonants. In the fourth step, multi-sequence alignment is performed. The output alignments of step 2 and 3 are combined based on their overlapping transcription: the target spelling for spelling error detection and the target reading for reading error detection. In this way, the PCU-segmentation of the realized spelling is deduced. In this final spelling alignment, spelling errors can be detected as PCUs that are spelled incorrectly. In the final alignment output of the reading error detection pipeline, reading errors can be detected as PCUs that are read incorrectly.

We found that the phonetic transcriptions provided by the G2P webservice were not always consistent. To overcome this problem, we made the following decisions. Words with a grapheme transcription ending in *-en* (e.g., *kijken* (to watch)) get a phonetic transcription ending in /@/ (schwa). In addition, we specified that words ending in the graphemes *-auw* or *-ouw* get the phonetic transcription /OU/ and not /OU w/. Finally, we decided to make no distinction between /x/ (unvoiced) and /G/ (voiced), since the G2P output is not consequent in making this decision, and Dutch speakers do not consistently pronounce them as either voiced or unvoiced either.

3.3. Annotation

Phoneme-PCU transparency The result of step two of the error detection algorithm was the alignment of target graphemes and phonemes, which resulted in a sequence of PCU-phoneme mappings the word consists of. In this step, we labeled each PCU-phoneme mapping as either sound pure (in case it occurred in Table 1), or not sound pure. When a target word only contained primary PCU-phoneme mappings, the word was annotated as sound pure. This annotation layer was called *SoundPure*.

Multiple attempts A frequently occurring phenomenon in initial readers is that they need multiple attempts to read the target word. In the aligned target graphemes, target phonemes and realized phonemes, this is visible as one or more insertions at the beginning of the alignment. Using a search function, we automatically annotated multiple attempts at the word level. The results were stored in the annotation layer *MultipleAttempts_Automatic*.

Reading error type The next step is to label each aligned target phoneme and realized phoneme with an error type: insertion (a phoneme was added), deletion (a target phoneme was not read), substitution (a target phoneme was substituted). This is done by comparing each target phoneme and realized phoneme pair one-by-one. After that, from these alignments on phoneme level, a classification at word level is computed: correct, insertion (only one phoneme was inserted in the complete word), deletion (only one phoneme was deleted in the complete word), substitution (only one phoneme was substituted in the complete word), multi (there are multiple phonemes inserted, deleted and/or substituted) and delWord (the complete word is not read). These classifications were saved in the categorical annotation layer *ErrorType_Automatic*.

3.4. Analysis 1: Validation

The spelling error detection algorithm was validated in an earlier study (Harmsen et al., 2021a). To validate the performance of the automatic reading error detection algorithm, one annotator manually annotated all readings from the selected DART data in two layers. The first layer contains for each target reading a boolean value which indicates whether there are multiple attempts or not (*MultipleAttempts_Manual*). The second layer, *ErrorType_Manual*, contains a categorical value for each reading. This value describes the reading error type using the following values: correct, insertion, deletion, substitution, multi and delWord. To validate the performance of the reading error detection and classification algorithm, we computed Matthew's Correlation Coefficient (MCC) between the automatic and manual annotation layers of *MultipleAttempts* and *ErrorType*. We used the MCC metric since our dataset is unbalanced, as it has more correct than incorrect readings. The MCC is proven to be more trustworthy than Cohen's Kappa on imbalanced datasets (Chicco et al., 2021).

3.5. Analysis 2: Application

From both the reading data set (phonetic transcriptions of 672 read words) and the spelling data set (digitized writings of 58,800 dictation words), we automatically selected the sound pure target words, using the annotation layer *SoundPure*. The sound pure target readings and spellings are listed in Table 2.

Corpus	Sound pure target words
BasiScript (N=9)	fee huis keus ligt lip monteur rijst schrik steil
DART (N=51)	bal blauw boomstam buik deuk dof flits fop gat geit hout jaap jong juicht keelpijn klets koen kous lach lift lijn lus markt meetlat melk mug muis muur nicht proost reis saus schoen schraal schrift schrik schroef schroot schuur specht spierkracht sportpark sterk stoep strik toch vang vorst vuur warmst zwart

Table 2: The sound pure target words from the DART reading tests and BasiScript dictations.

The spelling and reading error detection pipeline yields for each realized spelling an alignment of the target graphemes, target phonemes and realized phonemes, and for each realized reading, an alignment of target graphemes, target phonemes and realized phonemes. From these two multi-sequence alignments, we extract a list of target PCUs that are spelled at least one time incorrectly, and a list of target PCUs that are read at least one time incorrectly. For each PCU in each list, we compute the following measures:

Number of different targets The number of different target words in which the target PCU occurs. If this number is small, the target words themselves are printed.

Number of realized writings/readings (N) How often the target words that contain the selected target PCU are spelled/read in the complete dataset.

Absolute incorrect count How often the target PCU was spelled/read incorrectly.

Relative incorrect percentage How often the target PCU was spelled/read incorrectly with respect to how often this target PCU had to be spelled/read in total.

Aligned realized PCUs/phonemes A list of incorrect realized PCUs (in case of spelling errors) or phonemes (in case of reading errors) of the target PCU. The list is sorted from most to least frequently occurring realization

4. Results

4.1. Validation of the reading error algorithm

We computed the agreement between *MultipleAttempts_Manual* and *MultipleAttempts_Automatic*

Corpus	Total	Correct	Incorrect
BasiScript	21168	15323 (72%)	5845 (28%)
DART	321	207 (64%)	114 (36%)

Table 3: The number of analyzed realized writings (BasiScript) and readings (DART) of sound pure target words, together with their classification as either correctly or incorrectly read/spelled.

and found MCC = 0.87. In addition, we computed the agreement between *ErrorType_Manual* and *ErrorType_Automatic* and we found MCC = 0.92. So, for both *MultipleAttempts* and *ErrorType*, we found a high agreement between the manually and automatically obtained values.

4.2. Application

4.2.1. Description of the data

Table 3 presents the number of times a sound pure target word is read or written in the two data sets. The sound pure target words from BasiScript are written 21168 times by 2352 different writers. 28% of these written words contain at least one error. The sound pure words from DART are read 321 times by 28 different readers. From these read words, 36% contains at least one reading error.

4.2.2. Frequently made errors

Table 4 and Table 5 present respectively all incorrectly spelled target PCUs and all incorrectly read target PCUs, together with measures computed from the multi-sequence alignments. The results in Table 4 and 5 are ordered in descending absolute incorrect count, and in the right column per row in descending percentage.

In Table 4, we can observe that the primary PCU *ei* that represents the phoneme /EI/ is the PCU that is most frequently written incorrectly, i.e. in more than 80% of the times that this phoneme had to be written. In almost half of the times (49.91%) it was misspelled as *ij* and in 24.0% as *e*. Four other PCUs with a relative error percentage higher than 10% are *eu* (substituted most often with *u*), *g* (substituted with *ch*), *ij* (substituted with *ei*) and *ch* (deleted). In addition, we see that the * appears relatively high in the left column of the table (representing an insertion of a PCU), since it occurred 439 times in the selected data. The PCU *e* is most often (48.3%) inserted, followed by the PCU *u*.

Table 5 presents all incorrectly read target PCUs. The target PCUs that are relatively most frequently read incorrectly are *v* (33.33% of 9 readings), *ui* (23.53% of 17 readings), *ng* (22.22% of 9 readings), *a* (18.06% of 72 readings) and *g* (15.79% of 9 readings). These PCUs have the highest relative incorrect percentage.

5. Discussion

In this paper, we have presented an extension of the automatic spelling error detection and classification algorithm (Harmsen et al., 2021a,b) that is capable of detecting reading errors in phonetic transcriptions of audio recordings, in such a way that they are comparable with spelling errors. For reading error detection, the inputs to the algorithm are the target spelling and a realized (incorrect) phonetic transcription. For spelling error detection the inputs are the target spelling and the realized (incorrect) spelling. The output of the joint algorithm consists both in the reading and writing condition of a PCU-segmentation of the target spelling. In addition, each target PCU from the PCU-segmentation is aligned with its target phoneme and has a marking indicating whether it was read or spelled correctly in the realized reading or spelling. In case of a reading error, the substituted or inserted phoneme is returned. In the case of a spelling error, the substituted or inserted PCU is returned.

To answer research question 1, the reading error detection algorithm was evaluated by comparing automatically detected reading errors with manually annotated reading errors in a selection of phonetic transcriptions of read words from the DART corpus. We found a high level of agreement between the automatic and the manual scores.

Next, we applied the reading and spelling error detection algorithm to analyze reading and spelling errors in a selection of sound pure words in two separate corpora, one containing read words by first graders and one containing written words by second graders. To answer research question 2a, we analyzed the realized writings. We found that the PCUs that are more often written incorrectly are *ei*, *eu*, *g*, *ij* and *ch*. We observed that *ij* and *ei* are often exchanged, most probably because they sound the same (as /EI/). In addition, these target PCUs were presented in the target words *steil* (steep) and *rijst* (rice), in which substitution of the *ei* or *ij* results in the words *stijl* (style) and *reist* ((he) travels), which are both existing Dutch words. Earlier studies have proven that these two aspects make spelling more difficult (Bosman and de Groot, 1996; van Assche et al., 2014), which explains the fact that children make this specific error in writing this word. The same two explanations seem to hold for the finding that *g* in the word *ligt* ((he) lays) was substituted by *ch* in almost one fourth of the cases it had to be written. *g* and *ch* correspond to the same phoneme (i.e. /x/) and *licht* (light) is also an existing word in Dutch.

The frequent misspelling of the PCU *eu* occurring in the words *monteur* (mechanic) and *keus* (choice) might have another explanation. The PCU *eu* (with primary phoneme /EU/) is a vowel

Target PCU	Target words	N	Incorrect Absolute (#)	Incorrect Relative (%)	Realized PCUs (%)
ei	steil	2342	1875	80.06	ij (49.91), e (24.0), *ei (19.94)*, ee (2.73), 11 others
eu	monteur, keus	4663	1401	30.05	eu (69.95), u (17.39), e (4.61) uu (3.0), ui (2.06), 17 others
g	ligt	2345	616	26.27	g (73.73), ch (24.48), 8 others
ij	rijst	2331	590	25.31	ij (74.69), ei (22.35), ie (1.29), ui (0.3), 18 others
*	9 different words	-	439	-	e (48.3), u (14.29), j (6.58), i (5.9), t (3.85), n (3.17), 19 others
t	rijst, steil, monteur, ligt	9362	314	3.35	t (96.65), * (1.52), d (1.04), dt (0.34) tt (0.12), h (0.1), 10 others
ch	schrik	2332	250	10.72	ch (89.28), * (7.59), g (2.02), 7 oth.
r	rijst, monteur, schrik	7007	225	3.21	r (96.79), * (2.88), l (0.1), 8 others
f	fee	2323	181	7.79	f (92.21), v (7.06), t (0.13), * (0.13), 9 others
s	schrik, steil, keus, huis, rijst	11640	144	1.24	s (98.76), * (0.69), z (0.44), 8 others
l	steil, lip, ligt	7029	81	1.15	l (98.85), * (0.53), j (0.17) b (0.11), s (0.1), 9 others
i	schrik, lip, ligt	7018	69	0.98	i (99.02), ie (0.3), * (0.2) ee (0.19), e (0.13), 5 others
ui	huis	2316	63	2.72	ui (97.28), i (2.07), 9 others
p	lip	2341	59	2.52	p (97.48), b (1.28), 7 others
ee	fee	2323	57	2.45	ee (97.55), e (1.68), 6 others
k	schrik, keus	4651	33	0.71	k (99.29), * (0.34), h (0.19), 6 oth.
o	monteur	2344	30	1.28	o (98.72), oo (0.51) * (0.3) a (0.26), 2 others
m	monteur	2344	8	0.34	m (99.66), 3 others
h	huis	2316	3	0.13	h (99.87), 2 others

Table 4: Results of the BasiScript spelling error analysis. For each target PCU that is written incorrectly at least one time, we present the measures described in Section 3.5. An asterisk in the first column represents an insertion of a PCU. An asterisk in the last column represents a deletion of the target PCU.

and is written using two graphemes: *e* and *u*. These graphemes figure in as many as nine other PCUs: *e*, *ee*, *ei*, *u*, *uu*, *ui*, *eu*, *ou*, *au* and *oe*, which might be confusing for initial writers. This explanation is supported by the fact that the PCUs that initial writers write instead of the *eu* are *u*, *e*, *uu* and *ui*, which all contain an *e* or *u*.

With respect to the analyzed readings (research question 2b), we observed that the PCUs *v*, *ui*, *ng*, *a* and *g* are relatively most frequently read incorrectly. For most of these cases, the absolute number of errors is rather small. The errors for *a* are probably caused by the following complex but systematic vowel distinction in Dutch. In Dutch there is a phonological distinction between tense ('long') and lax ('short') vowels, as in the case of *oo* and *o*. In Dutch, letter doubling is used to distinguish between tense and lax vowel phonemes in monosyllables. The lax vowel *o* is written with one letter *o*, but the tense vowel *oo* is written with a single letter in open syllables (*bo-men (*=trees)) and with double letters *oo* in closed syllables (*boom (= tree*)) . This distinction can of course be confusing for beginning learners.

To answer research question 2c, we investigated the overlap between spelling and reading errors. This overlap seems to be limited. We found that *ch* is often deleted, both in spelling and reading, and that a *g* is often substituted by *ch* and /g/ respectively. However, there seem to be no clear reasons that can explain these specific errors. The limited overlap between spelling and reading errors could be explained by the fact that Dutch has a higher feedforward consistency than feedback consistency (Bosman and Van Orden, 1997). However, since several variables were not controlled for (i.e., the target words that had to be read and spelled, the participants, and the size of the datasets), we are not able to make a strong claim on this aspect. However, the joint spelling and reading error analysis method we presented in this study enables further research in this direction.

Target PCU	# Diff. targets	N	Incorrect Absolute (#)	Incorrect Relative (%)	Realized phonemes (%)
*	28	-	208	-	s (12.02), t (10.58), x (7.69), @ (7.21) r (6.73), p (6.25), k (4.81), , 25 others
t	24	175	17	9.71	t (90.29), * (4.0), s (2.29), l (1.71), f (0.57) k (0.57), p (0.57)
r	18	139	14	10.07	r (89.93), * (5.76), l (2.16), x (0.72), d (0.72), w (0.72)
a	11	72	13	18.06	A (81.9), a (12.5), O (2.8), E (1.4), e (1.4)
l	12	80	10	12.5	l (87.5), r (3.75), * (2.5), p (1.25), d (1.25), k (1.25), m (1.25), h (1.25)
ch	13	86	8	9.3	x (90.7), * (4.65), k (1.16), r (1.16), N (1.16), h (1.16)
s	24	152	7	4.61	s (95.39), t (0.66), p (0.66), S (0.66), b (0.66), z (0.66), * (0.66), v (0.66)
k	13	86	6	6.98	k (93.0), * (3.5), r (1.2), p (1.2), t (1.2)
i	6	43	4	9.3	I (90.7), i (4.65), y (2.33), u (2.33)
ui	3	17	4	23.53	UI (76.47), U (11.76), EU (5.88), I (5.88)
o	6	34	4	11.76	O (88.24), o (5.88), A (2.94), UI (2.94)
g	3	19	3	15.79	x (84.21), g (10.53), S (5.26)
v	3	9	3	33.33	v (66.67), f (22.22), s (11.11)
m	8	63	3	4.76	m (95.24), b (1.59), h (1.59), p (1.59)
p	9	63	3	4.76	p (95.24), r (1.59), * (1.59), b (1.59)
ei	2	18	2	11.11	EI (88.89), UI (5.56), i (5.56)
ng	2	9	2	22.22	N (77.78), x (11.11), n (11.11)
f	6	44	2	4.55	f (95.45), * (4.55)
n	5	21	2	9.52	n (90.48), l (4.76), * (4.76)
ee	2	14	1	7.14	e (92.86), @ (7.14)
uu	3	16	1	6.25	y (93.75), U (6.25)
oo	3	16	1	6.25	o (93.75), AU (6.25)
ou	2	5	1	20.0	AU (80.0), UI (20.0)
e	4	25	1	4.0	E (96.0), @ (4.0)
b	4	24	1	4.17	b (95.83), d (4.17)

Table 5: Results of the DART reading error analysis. For each target PCU that is read incorrectly at least one time, we present the measures described in Section 3.5. An asterisk in the first column represents an insertion of a PCU. An asterisk in the last column represents a deletion of the target PCU.

6. Future Directions

In the introduction to this paper we made clear that this is only the beginning of a research endeavour that will certainly require more suitable data and optimized algorithms. The present study has indeed some limitations that were imposed by the complexity of the task and the scarcity of available data. For a first attempt, we decided to constrain ourselves to analyzing the so-called sound pure words. In a following step, we would like to extend this approach to other, more complex words, but then it is clear that we are likely to get an explosion of phoneme-grapheme mappings that will require a more complex classification scheme. This classification scheme should capture some characteristics that are specific for the speech of initial readers, like multiple attempts to read a word or insertion of vowels (Harmsen et al., 2023). In addition, to gain insight into specific individual difficulties, we would like to study reading and spelling data of one and the same child, preferably longitudinal data, in which children read and write the same words. So one important task for future research could be the collection of reading and spelling data of the same children. Another future direction is to use Automatic Speech Recognition (ASR) to automatically obtain phonetic transcriptions of child read speech. Currently, ASR models for automatic word correctness assessment are available for Dutch (e.g., Molenaar et al. (2023); Harmsen et al. (2023)), but a well performing and evaluated ASR model for phoneme recognition in child read speech has not yet been published. Such a model could be inspired by research by Gelin et al. (2023), who have recently published about developing such models for automatic phoneme recognition in French.

7. Ethical statement

The present research and its results may have a major societal impact as they contribute to significantly increasing the reliability and validity of reading and writing assessment and ultimately paving the way to improving and personalizing learning-to-read-and-write trajectories.

8. Acknowledgements

We would like to thank Stéphanie Kremer for making phonetic transcriptions of the child audio recordings. This publication is part of the ASTLA project with project number 406.20.TW.009, which is (partly) financed by the Dutch Research Council (NWO).

9. Bibliographical References

Y. Bai, F. Hubers, C. Cucchiarini, and H. Strik. 2021. An ASR-based reading tutor for practicing reading skills in the first grade: Improving performance through threshold adjustment. In *Proc. IberSPEECH 2021*, pages 11–15.

Y. Bai, F. Hubers, C. Cucchiarini, R. van Hout, and H. Strik. 2022. The effects of implicit and explicit feedback in an ASR-based reading tutor for Dutch first-graders. In *Proc. Interspeech 2022*, pages 4476–4480.

S.R. Borgwaldt, F. Hellwig, and A.M.B. De Groot. 2004. Word-initial entropy in five languages: Letter to sound and sound to letter. *Written Language and Literacy*, 7:165–184.

A. M. T. Bosman and A. de Groot. 1996. Phonologic mediation is fundamental to reading: Evidence from beginning readers. *The Quarterly Journal of Experimental Psychology Section A*, 49(3):715–744.

A. M. T. Bosman and G. C. Van Orden. 1997. *Why Spelling is More Difficult than Reading*, pages 173–194. Lawrence Erlbaum Associates Publishers, Mahwah, NJ, US.

D. Chicco, M.J. Warrens, and G. Jurman. 2021. The Matthews Correlation Coefficient (MCC) is more informative than Cohen's Kappa and Brier Score in binary classification assessment. *IEEE Access*, 9:78368–78381.

L. Cleuren, J. Duchateau, P. Ghesquière, and H. Van Hamme. 2008. Children's oral reading corpus (CHOREC): Description and assessment of annotator agreement. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA).

C. Cucchiarini. 1993. *Phonetic transcription: A methodological and empirical study*. Ph.D. thesis, Nijmegen, The Netherlands.

C. Cucchiarini. 1996. Assessing transcription agreement: Methodological aspects. *Clinical Linguistics and Phonetics*, 10:131–155.

C. Cucchiarini, H. van Hamme, O. van Herwijnen, and F. Smits. 2006. JASMIN-CGN: Extension of the Spoken Dutch Corpus with speech of elderly people, children and non-natives in the human-machine interaction modality. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy. European Language Resources Association (ELRA).

L. Gelin, M. Daniel, T. Pellegrini, and J. Pinquier. 2023. Comparing phoneme recognition systems on the detection and diagnosis of reading mistakes for young children's oral reading evaluation. In *Proc. 9th Workshop on Speech and Language Technology in Education (SLaTE)*, pages 6–10.

W. Harmsen, C. Cucchiarini, and H. Strik. 2021a. Automatic detection and annotation of spelling errors and orthographic properties in the Dutch BasiScript corpus. *Computational Linguistics in the Netherlands Journal*, 11:281–306.

W. Harmsen, C. Cucchiarini, and H. Strik. 2021b. Automatic quantitative analysis of spelling errors in texts written by sixth graders. In *EDULEARN21 Proceedings*, 13th International Conference on Education and New Learning Technologies, pages 8937–8945. IATED.

W. Harmsen, F. Hubers, R. van Hout, C. Cucchiarini, and H. Strik. 2023. Measuring word correctness in young initial readers: Comparing assessments from teachers, phoneticians, and ASR models. In *Proc. 9th Workshop on Speech and Language Technology in Education (SLaTE)*, pages 11–15.

J. Keuning and L. Verhoeven. 2008. Spelling development throughout the elementary grades: The Dutch case. *Learning and Individual Differences*, 18(4):459–470.

M.H.L. Kleijnen. 1997. *Strategieën van zwakke lezers en spellers in het voorgezet onderwijs*. Ph.D. thesis, Vrije Universiteit Amsterdam, Lisse.

R. Laarmann-Quante. 2016. Automating multi-level annotations of orthographic properties of German words and children's spelling errors. In *Proc. Language Teaching, Learning and Technology (LTLT 2016)*, pages 14–22.

K. Landerl and P. Reitsma. 2005. Phonological and morphological consistency in the acquisition of vowel duration spelling in Dutch and German. *Journal of Experimental Child Psychology*, 92(4):322–344.

S. Limonard, C. Cucchiarini, R.W.N.M. van Hout, and H. Strik. 2020. Analyzing read aloud speech by primary school pupils: Insights for research and development. In *Proc. Interspeech 2020*, pages 3710–3714.

B. Lin and L. Wang. 2022. Phoneme mispronunciation detection by jointly learning to align. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6822–6826.

B. Molenaar, C. Tejedor-Garcia, C. Cucchiarini, and H. Strik. 2023. Automatic assessment of oral reading accuracy for reading diagnostics. In *Proc. Interspeech 2023*, pages 5232–5236.

A. Nunn. 1998. Dutch orthography: A systematic investigation of the spelling of Dutch words. Holland Academic Graphics, Den Haag.

A. Tellings, N. Oostdijk, I. Monster, F. Grootjen, and A. van den Bosch. 2018a. BasiScript: : A corpus of contemporary Dutch texts written by primary school children. *International Journal of Corpus Linguistics*, 23(4):494–508.

A. Tellings, N. Oostdijk, I. Monster, F. Grootjen, and A. van den Bosch. 2018b. Spelling errors of 24 cohorts of children across primary school 2012-2015: A BasiScript corpus study. *Computational Linguistics in the Netherlands Journal*, 8:83–98.

E. van Assche, W. Duyck, and R.J. Hartsuiker. 2014. Phonological recoding in error detection: A cross-sectional study in beginning readers of Dutch. *PLOS ONE*, 8(12).

M. van Druenen, M. Gijsel, F. Scheltinga, and L. Verhoeven. 2019. *Leesproblemen en dyslexie in het basisonderwijs: Handreiking voor aankomende leerkrachten*, 3rd edition. Masterplan Dyslexie Expertisecentrum Nederlands, 's-Hertogenbosch.

Z. Zhang, Y. Wang, and J. Yang. 2021. Text-conditioned transformer for automatic pronunciation error detection. *Speech Communication*, 130:55–63.

10. Language Resource References

Bai et al. 2020. *Dutch Automatic Reading Tutor (DART) Corpus*. Radboud University.

Elffers et al. 2013. *ADAPT: Algorithm for Dynamic Alignment of Phonetic Transcriptions*. Radboud University. [link].

Gillis. 2001. *Protocol for Broad Phonetic Transcriptions*. Corpus Gesproken Nederlands (CGN) Project. [link].

Tellings et. al. 2015. *BasiScript-corpus*. Radboud University. Dutch Language Institute, 1.0. [link].

Ten Bosch. 2019. *Grapheme to Phoneme Converter*. Centre for Language and Speech Technology, 0.3.4. [link].

Tool for Constructing a Large-Scale Corpus of Code Comments and Other Source Code Annotations

Luna Peck, Susan Windisch Brown
University of Colorado Boulder
{luna.peck, susan.brown}@colorado.edu

Abstract

The sublanguage of source code annotations—explanatory natural language writing that accompanies programming source code—is little-studied in linguistics. To facilitate research into this domain, we have developed a program prototype that can extract code comments and changelogs (i.e. commit messages) from public, open-source code repositories, with automatic tokenization and part-of-speech tagging on the extracted text. The program can also automatically detect and discard "commented-out" source code in data from Python repositories, to prevent it from polluting the corpus, demonstrating that such sanitization is likely feasible for other programming languages as well. With the current tool, we have produced a 6-million word corpus of English-language comments extracted from three different programming languages: Python, C, and C++.

Keywords: text extraction, corpora, code comments

1. Introduction

Within the already-dizzying number of human languages spoken around the world, one can find an even greater number of sublanguages: niche subsets of a language that emerge for specialized purposes. While some sublanguages, such as cooking recipes (e.g. Brdar-Szabó and Brdar, 2009; Gerhardt et al., 2013; DiMeo and Pennell, 2018), have received widespread linguistic attention, others, such as source code annotations, remain woefully understudied.

To the end of eventually developing a large, well-curated corpus of source code annotations, we have developed a tool[1] for automatically extracting such data from source code repositories. By "source code annotation," we are referring to explanatory natural language writing that accompanies formal language programming source code. Examples include comments, changelogs (commit messages)[2], and documentation.

The current version of the program builds a corpus of approximately 6-million words, across 90,461 changelogs and 76,445 comments, taken from three English-language repositories written in the program's supported programming languages: Python (django[3]), C (libvirt[4]), and C++ (dlib[5]). These repositories were chosen as they host code for well-established, widely-used software implemented in the languages currently supported by the program. The program automatically performs tokenization and part-of-speech tagging on the extracted data. The program does not currently specially extract documentation, and documentation that is encoded as comments (e.g., Doxygen) is not differentiated from other comments.

We have discovered that code comments in particular are a very messy domain, requiring much sanitization, such as:

1. Resolving inconsistent encoding and stripping invalid characters (e.g., a changelog contained a bell character, unicode 0x07).

2. Identifying and discarding source code that has been disabled by "commenting it out."

3. Identifying snippets of code embedded in natural language comments and preprocessing them before annotation (e.g., part-of-speech tagging).

The program currently performs (1)[6] and (2) automatically[7]. (3) will be handled in a future version of the program. Due to space constraints, we have only devoted significant discussion to how we have

[1] https://github.com/lunaria-bee/ccc-tools/tree/cawl2024-docs

[2] "Changelog" is a generic term we have chosen to refer to any description of changes made to the source code. In the current version of the tool and its resulting corpus, "changelog" is synonymous with "commit message." However, some codebases, such as those that are not under version control, may log changes in other ways, e.g. simply with comments. The use of "changelog" anticipates a possible future version of this tool that extracts data from such repositories.

[3] https://github.com/django/django

[4] https://github.com/libvirt/libvirt

[5] https://github.com/davisking/dlib

[6] Although we are handling this somewhat reactively, with special handling added for each new inconsistency as it is encountered.

[7] Although (2) currently only functions for comments in Python code.

approached (3), as we consider it the most interesting of the challenges to which we have some solutions.

2. Motivations and Prior Work

This project was inspired by previous (unpublished) work in which we made basic analyses of source code annotations from a small, largely hand-collected dataset. While the smaller dataset was adequate for primarily qualitative analysis, substantive statistical linguistic analysis demands larger datasets.

Source code annotations are particularly interesting in that they exist almost purely in written form. While they interact with the often-spoken sublanguage of software development jargon, comments themselves are both constructed and interpreted overwhelmingly as written language. Further, source code annotations offer an unusual opportunity to study the ways natural language interacts with formal languages: How does writing about formal language influence our production choices in natural language? Are such influences consistent or varied across different formal + natural language pairings?

There is little existing purely linguistic research into this domain. Much existing work has focused on instrumental tasks, like automatically generating comments from source code (Song et al., 2019), automatically generating changelog messages from source code (e.g. Cortés-Coy et al., 2014), and automatically generating source code from natural language prompts (e.g. Barone and Sennrich, 2017; Wei et al., 2019). Even for such instrumental purposes, having a linguistic corpus of source code annotations should prove valuable; the better we understand how programmers write about code, the better we can create tools to translate between natural language and source code.

The small amount of purely linguistic research on the topic has primarily involved Letha Etzkorn (Etzkorn et al., 2001; Vinz and Etzkorn, 2008). This research studied the linguistics of code comments in 11 C++ packages, as opposed to this project, which aims to construct a corpus of multiple annotation types extracted from multiple different programming languages. To our knowledge, the corpus used in Etzkorn et al.'s research has not been published.

A similar project to automatically extract source code annotations from source repositories has been conducted (Barone and Sennrich, 2017), although that project was much more limited in scope than this current project. The Barone & Sennrich project focused only on extracting docstrings[8] from Python programs. This current project, on the other hand, aims to extract a broad range of annotations from source code, including comments, changelogs, and eventually documentation.

Automatically filtering comments for linguistic use has been previously explored (Matskevich and Gordon, 2022). Similarly to Barone and Sennrich (2017), this research focused only on comments, excluding other forms of source code annotations. Nonetheless, the preprocessing techniques described by Matskevich and Gordon (2022) will almost certainly be useful for data sanitization in future versions of this tool.

3. Data Structure

Corpus data is stored as XML. Each source code annotation, be it a changelog or comment, is generically referred to as a "note." The XML structure for each corpus file contains a root node, named `<notes>`, that in turn contains a series of `<note>` elements, each representing a single source code annotation.

Each <note> element contains the raw text of the annotation, its tokenization and part-of-speech tags, and assorted metadata, represented by the following subelements:

- `<author>`: First 8 bytes of the SHA-256 hash of the author's version control username. When an annotation has multiple authors, there is one `author` subelement per author.
- `<repo>`: Name of the repository the data came from.
- `<revision>`: First 7 nibbles[9] of the revision hash, as provided by Git. The first 7 nibbles of the hash is the same ID scheme used by GitHub. When an annotation was edited across multiple revisions, there is one `revision` subelement per revision.
- `<note-type>`: One of 'changelog' or 'comment'.
- `<language>`: Programming language the data was extracted from. Only applicable for comments, not changelogs.
- `<file>`, `<first-line>`, & `<last-line>`: Path to the file a note was extracted from, and the span of lines within that file on which it appears. Only applicable for comments, not changelogs. Facilitates finding the source code associated with a comment.
- `<raw>`: Raw text of the annotation, in which comment delimiters (if applicable) and newlines & other whitespace are preserved.

[8] In-source documentation, see https://peps.python.org/pep-0257/.

[9] A half-byte, i.e. four bits.'

```
<note>
  <repo>dlib</repo>
  <author>0e8171ad9c374e5d</author>
  <revision>754da0e</revision>
  <note-type>comment</note-type>
  <file>repos/dlib/dlib/algs.h</file>
  <first-line>205</first-line><last-line>206</last-line>
  <language>c</language><
  raw>// set the initial guess for what the root is depending on
// how big value is
</raw><
  tokens>set the initial guess for what the root is depending on how big value is
   </tokens>
  <pos>VB DT JJ NN IN WP DT NN VBZ VBG IN WRB JJ NN VBZ</pos>
</note>
```

Figure 1: An example `<note>` element.

- `<tokens>`: Tokenized text. Comment delimiters (if applicable) and original whitespace are stripped from the data. Spaces represent word token separators, and newlines represent sentence token separators.

- `<pos>`: Part-of-speech annotations aligned to the tokenized text. Like in the <tokens> subelement, spaces represent word token separators, and newlines represent sentence token separators. The tags are those assigned by NLTK's `nltk.tag.pos_tag()` function (Bird et al., 2009). A future version of the program may include a part-of-speech tagger designed for the specific domain of source code annotations.

4. Data Extraction

4.1. Comments

The program checks each file in each repository against parsers[10] for each supported programming language. If a file validates as a valid source file in a supported language, the contents of the file will be tokenized by an appropriate parser[11]. If two or more comment tokens appear across consecutive lines, those tokens are grouped into a single comment.

[10] Python's built-in `ast.parse()` method for Python, libclang for C and C++.

		Actual	
		Natural	Code
Predicted	Natural	50	0
	Code	40	10

Figure 2: Confusion matrix showing efficacy of our approach to identifying commented-out Python code on random samples of 50 comments predicted to be natural language and 50 comments predicted to be code. Taking Natural Language (i.e. inclusion in the corpus) as the positive label: Precision=1.0, Recall=0.56.

As the purpose of this corpus is to study natural language text, programming code that has been disabled by "commenting it out" should not be included in the dataset. Automated detection of commented-out source code proved somewhat tricky. It is not sufficient to discard any comment that could be interpreted as valid code, as there are many possible natural language comments that look like source code to the right parser. For example, all of the following are valid Python code:

- Single words: e.g., *deprecated*
- A word followed by another word in parentheses: e.g., *Tests (Final)*
- Words separated by mathematical operators like +, -, *, or /: e.g., *Pre-increment/decrement*

Naïvely discarding any comment that parses as programming code risks removing valuable data from the corpus, and so a smarter approach is in order.

After experimenting with several different approaches, for Python we settled on the following policy: A comment is discarded as commented-out code if (1) its contents are valid Python code and (2) it contains parentheses, square brackets, equals signs, periods, or the word "return".

These rules work quite well, discarding:

- Function calls and object construction: e.g., *Color(0, 0.56789, 0, .5)*.

[11] Python's built-in `tokenize` library for Python, libclang for C and C++.

```
Identified using the following command:

$ git grep -I '\(\<[_a-zA-Z0-9]\+\>\) *= *\1 *[-+/*^%&|<>@]'
```

Figure 3: Bash snippet with git command embedded in a changelog.

- Indexing: e.g., *text[col-1]*.
- Assignments: e.g., *ay += node.y*.
- Subscripting: e.g., *self._trigger_layout*.
- Return statements: e.g., *return None*.

...while keeping many natural language comments that coincidentally resemble source code:

- *Initialize*
- *todo: remove*
- *--Save/Cancel*

This approach does sometimes discard natural language comments we would probably want to keep, such as *return everything in strings*, discarded due to the presence of the keyword "*return*", causing this to resemble an instruction to return whether `everything` is present in some collection `strings`. Although discarding such data is unfortunate, we have decided to prioritize precision over recall in this task, as we deem polluting the dataset with source code more damaging than discarding some potentially useful data. This over-discarding could likely be ameliorated by augmenting these symbolic rules with a learned model that classifies strings into code and non-code.

The program currently does not attempt to identify commented-out C/C++ code. libclang is not designed for parsing fragments of C/C++, expecting to produce a full translation unit from any input. One option for overcoming this limitation would be to attempt to parse the AST output by libclang[12] against a simplified C++ grammar. Another option would be to rely entirely on a learned model for identifying commented-out C/C++ code.

Author and revision information for each comment are retrieved using the `git blame` command, which provides the most-recent commit modifying each line of a file.

4.2. Changelogs

Currently, all data included in the corpus comes from Git repositories, so the program can trivially extract revision information from each repository's commit history. The raw text of the changelog note is simply the commit message, the author is the author of the commit, and the revision is the revision created by the commit.

5. NLTK Reader

The demonstration corpus may be explored using the CccReader() class, an extension of NLTK's CategorizedCorpusReader and XMLCorpusReader classes (Bird et al., 2009). It provides the general functions one expects of an NLTK reader: words(), sents(), etc.

6. Future Work

6.1. Handling Code Snippets

While comments that consist entirely of source code should be removed from the corpus, comments and changelogs that include snippets of code embedded within natural language text (often for illustrative purposes) are quite common and should be included in the corpus. However, these source code snippets tend to confuse automated annotation functions (such as those responsible for part-of-speech tagging), as natural language concepts such as part of speech do not map cleanly to source code. Therefore, such snippets must be identified and given special handling.

Identifying code snippets in source annotations is much more challenging than identifying commented-out code, for two main reasons:

- As commented-out code is discarded, and discarding some potentially-useful data is more acceptable than polluting the corpus with source code, that algorithm can simply err on the side of over-discarding. This algorithm, however, would need to identify source code snippets in data that has already been included in the corpus, demanding much higher accuracy.
- The algorithm for discarding commented-out code examines each comment in its entirety. Identifying code snippets, however, would require identifying all spans of text that should be considered source code. Thus, the algorithm

[12]libclang's `parse()` method produces an AST even for invalid input.

doesn't have to be run against one string, but many substrings of each source annotation.

- Source code snippets are not guaranteed to be written in the same language as the surrounding source code. Consider the Bash snippet in Figure 3.

6.2. Tracking Repositories Across Time

Currently, the program takes a snapshot of each repository at a particular revision level, to ensure that all users invoking the program receive the same corpus as output. However, tracking repositories across multiple revisions could potentially provide useful data regarding how programmers revise comments as code evolves, and allow diachronic analysis of source code annotations.

7. Conclusions

Based on the work completed here, it seems quite feasible to build a corpus of source code annotations much larger than the 6-million-word corpus produced at this time. Although not all of the necessary work can be automated, in our opinion the task is automatable enough to justify pursuing the project further. We intend to continue developing this project, with the goal of building a larger, more diverse corpus of source code annotations, to further develop our understanding of this little-studied domain of human language.

8. Bibliographical References

Antonio Valerio Miceli Barone and Rico Sennrich. 2017. A parallel corpus of python functions and documentation strings for automated code documentation and code generation. ArXiv.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Rita Brdar-Szabó and Mario Brdar. 2009. Indirect directives in recipes: a cross-linguitic perspective. *Lodz Papers in Pragmatics*, 5(1):107–131.

Luis Fernando Cortés-Coy, Mario Linares-Vásques, Jairo Aponte, and Denys Poshyvaynk. 2014. On automatically generating commit messages via summarization of source code changes. In *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*, Victoria, BC, Canada.

Michelle DiMeo and Sara Pennell. 2018. *Reading and writing recipe books, 1550–1800:*. Manchester University Press, Manchester, England.

Letha H. Etzkorn, Carl G. Davis, and Lisa L. Bowen. 2001. The language of comments in computer software: A sublanguage of english. *Journal of Pragmatics*, 33(11):1731–1756.

Cornelia Gerhardt, Maximiliane Frobenius, and Susanne Ley, editors. 2013. *Culinary Linguistics*. John Benjamins.

Sergey Matskevich and Colin S. Gordon. 2022. Preprocessing source code comments for linguistic models.

Xiaotao Song, Hailong Sun, Xu Wang, and Jiafei Yan. 2019. A survey of automatic generation of source code comments: Algorithms and techniques. *IEEE Access*, 7:111411–111428.

Bradley Vinz and Letha Etzkorn. 2008. Comments as a sublanguage: A study of comment grammar and purpose. In *Proceedings of the 2008 International Conference on Software Engineering Research and Practice, SERP 2008*, pages 17–23.

Bolin Wei, Ge Li, Xin Xia, Zhiyi Fu, and Zhi Jin. 2019. Code generation as a dual task of code summarization.

Tokenization via Language Modeling: the Role of Preceding Text

Rastislav Hronsky, Emmanuel Keuleers
Jheronimus Academy of Data Science, Tilburg University
Sint Janssingel 92 5211 DA ’s-Hertogenbosch, Warandelaan 2 5037 AB Tilburg
r.hronsky@tue.nl, E.A.Keuleers@tilburguniversity.edu

Abstract

While language models benefit immensely from their capacity to model large context (i.e., sequence of preceding tokens), the role of context is unclear in text tokenization, which is, in many cases, language model-driven to begin with. In this paper, we attempt to explore the role in three different writing systems and using three different text tokenization strategies (word-based, Morfessor, and BPE). In the first experiment, we examined how the size of context used for predicting the next token affects the ranking of the segmentation strategies i.t.o. language model surprisal. This effect was very writing system specific: minimal in case of English, and rank-reversing due to increased context size and token granularity in case of Turkish and Chinese. In the second experiment, we examined how context alters segmentation hypotheses when using language models to identify word boundaries. In this case, the effect was subtle: using context-aware, rather than context-free segment scores improved boundary recognition accuracy by up to 0.5%, once baseline effects were exploited.

Keywords: language modeling, tokenization, word segmentation

1. Introduction

The most basic unit of computer-stored written language is typically the character. Despite that neural network based systems are capable of taking characters as input, it is still common practice to divide the signal into linguistically more meaningful chunks (i.e., tokens). Most writing systems include conventions, such as whitespace and punctuation, that can help with segmentation. However, relying on these conventions to tokenize text is fragile: (1) there are many writing systems with different conventions, (2) even if explicit cues for word separation are available, further division, for instance of compound words, remains problematic, and (3) the noisiness and openness of natural language make dictionary-based string matching unreliable.

Therefore, modern systems pre-process text via pipelines that, in addition to using manually described rules, employ statistical segmentation, which is robust, language independent, and data driven. The idea is similar to how speech segmentation is described in studies on spoken language acquisition: a standalone token (word) is one where, within the boundaries, the regularity (mutual information) between neighboring elements (phonemes) is disproportionately stronger than at the boundaries (Saffran et al., 1996a).

This principle can be formulated more generally as a search for the set of segments that, by scoring the probability of each segment in isolation, maximizes the sequence generation probability (discounting any between-segment dependencies). In this form, it has been adopted as the decoding strategy for many text tokenization implementations (Creutz and Lagus, 2005; Virpioja et al., 2013; Sennrich et al., 2016; Kudo, 2018; Kudo and Richardson, 2018).

While these systems produce satisfactory tokens for their intended purposes, there is a lack of attention to the role of context in tokenization in natural language processing research. This is surprising, because statistical text segmentation is an application of probabilistic language models and modern language models have a capacity to model context extending to thousands of preceding tokens.

As a simple example of how context can affect segmentation, consider the sequence ’ishe’. Given a unigram model, the segmentation ’i-she’ may be optimal because ’i’ is a high frequency token in English. However, since ’is-he’ co-occurs often, the increased probability of ’he’ in the context of ’is’ may result in an overall higher probability of the ’is-he’ segmentation according to a bigram model. This way, it is conceivable how such a context-free (unigram) and context-sensitive (bigram) approach to segmentation would be in disagreement with each other.

Research on language acquisition confirms that context can affect segmentation. Language learners who make the independence assumption, hence ignore context, tend to identify words less accurately than ones that include the dependency to the preceding word (Goldwater et al., 2009).

A further indication of context utility comes from word segmentation research on writing systems without explicit word boundary notation: several systems employ contextual features to improve word segmentation performance (Meknawin, 1995; Kudo, 2006; Huor et al., 2004a; Durrani and Hussain, 2010).

To the best of our knowledge, the extent to which unigram and higher-order n-gram segmentation models correctly recover linguistic units, e.g., words, has not been studied in detail. Our first research question is:

(1) *How does using a higher-order language model (i.e., bigram or trigram as opposed to unigram) affect the performance of statistical word segmentation?*

To answer this question, we simulate word segmentation by deleting explicit word boundary notation and testing how well a uni-, bi-, and trigram model re-discover the reference word boundaries. We discuss the effects of increased n-gram model order in the context of merely using a more representative language model (i.e., inferred using a larger body of text), a baseline effect.

Given a particular segmentation of a corpus, we can derive a language model based on it and compute the average language model surprisal, a metric reflecting segmentation optimality. This is a common way to intrinsically assess segmentation, both between competing segmentation algorithms and within the decoding process of a single segmentation method, and it is the basis for our second, more general research question:

(2) *How does changing the order of the language model change the assessment of surprisal-based segmentation optimality?*

To answer this question, we present simulations examining the extent to which a particular corpus segmentation, e.g., a reference word segmentation, Morfessor, or BPE segmentation, ranks as consistently optimal (i.t.o. bits-per-sentence) across language models set to include increasingly long dependencies. If the ranking stays constant, this indicates a weak role of context.

We conducted both experiments with text in English (a morphologically poor language), Turkish (a very agglutinative language), and Chinese (a language with a logographic writing system). The corpus used was based on movie subtitles and aligned such that for each language it contains subtitles for the same set of movies.

One difficulty that we faced during this research was the lack of accessible scientific libraries to perform higher order segmentation. Therefore, we describe the process in Section 3, and also publicly release the code as a Python package [1] that we used to solve first, second, and third order sequence segmentation problems. The package is built on top of NetworkX (Hagberg et al., 2008), a popular network analysis library, providing direct access to a wide range of tools that can be used to manipulate and visualize the segmentation problems.

[1] https://github.com/hrasto/segmentgraph

2. Related Work

2.1. Word Recognition in Spoken Language

While certain acoustic features help with word segmentation in speech recognition (Jusczyk et al., 1993; Mattys et al., 1999), the exact mechanics are non-trivial: the signal is often noisy and contains hardly any explicit word boundary signature (Cole et al., 1980; Reddy, 1976). Tasking humans with word identification from spectrograms of continuous speech is problematic itself (Klatt and Stevens, 1973). Unsurprisingly, modern automatic speech recognition (ASR) omit manual feature engineering and learn to transcribe speech to words end-to-end (Anusuya and Katti, 2009; Hannun et al., 2014; Amodei et al., 2016).

The challenges associated with ASR naturally transfer to language acquisition research: how do language learners identify words? A prominent finding from this literature is that the expectation of a phoneme pair at a word boundary to have a lower transition probability than one within a word is a reliable cue for word segmentation (Saffran et al., 1996a,b). As a result, the idea of exploiting statistical properties to segment speech into words has gained prominence (Brent, 1999; Venkataraman, 2001; Batchelder, 2002). Relatedly, computational models capitalizing on regularities *between* words, in addition to within words, improve word boundary recognition (Goldwater et al., 2009), especially by reducing undersegmentation (falsely omitting word boundaries).

2.2. Writing Systems without Whitespace

Word segmentation is an important topic for languages employing writing systems without explicit word delimiters (e.g., Chinese, Japanese, Thai or Khmer). Using word units was mainly needed for efficient functioning of traditional information retrieval systems (Nie et al., 1996; Chen et al., 1997; Leong and Zhou, 1997; Foo and Li, 2004). Simplifying matters somewhat, the word segmentation methods related to statistical segmentation were based on: (1) variants of *dictionary* based string matching for Chinese (Chen and Liu, 1992; Sproat and Emerson, 2003), Thai (Rarunrom, 1991; Virach, 1993), Khmer (Bi and Taing, 2014a), Japanese (Sato, 1999); (2) *statistical* approaches for Chinese (Sproat and Shih, 1990; Ge et al., 1999; Sun et al., 1998), Thai (Pornprasertkul, 1994; Meknawin, 1995), Khmer (Huor et al., 2004a), Japanese (Matsumoto et al., 2000; Kudo, 2006); (3) pipelines involving the statistics and several other rules and features (Meknavin et al., 1997). However, recent research questions the necessity for word segmentation by arguing that modern models based on characters, instead of words, generalize better and reduce overfitting (Li et al., 2019).

2.3. Vocabularies in modern NLP

In English-centric research, the traditional unit – word or lemma – was just about rejected in favor of subwords once neural networks became mainstream (Mikolov et al., 2012). This shift was mainly motivated by conveniences such as reduction of vocabulary size and robustness in handling out-of-vocabulary situations. Discounting linguistic rigor and aiming for robust engineering, several algorithms were developed to segment text into short subword units. The methods were typically based on a greedy compression algorithm: byte-pair encoding (BPE) (Gage, 1994; Sennrich et al., 2016), and its derivatives (Schuster and Nakajima, 2012; Kudo and Richardson, 2018).

Several studies further examined the effects of segmentation on language modeling and related tasks. Huck et al. (2017) found that using linguistically informed segmentation (e.g., compound splitting, prefix splitting, etc.) can improve machine translation (MT) performance over purely compression-based segmentation. Domingo et al. (2019) concluded that, while segmentation affects MT performance, there is no clear winner in terms of algorithms, as performance varies across language pairs. In language modeling experiments, Liu et al. (2019) found that there was a small advantage in using BPE-derived tokens from characters rather than bytes, and Gallé (2019) report that tokenizers producing fewer (thus longer) segments perform better.

Lastly, research suggests that there are advantages in using morphologically aligned subwords. Bostrom and Durrett (2020) compared segmentation produced by BPE to the Unigram method (Kudo, 2018), and found the latter to produce more morpheme-like tokens and ultimately outperforming BPE. Similarly, Park et al. (2021) report advantages in using segmentations produced by Morfessor (Creutz and Lagus, 2005), an unsupervised morphological segmentation system, over the BPE-based segmentations. Both methods, Unigram (Kudo, 2018) and Morfessor (Creutz and Lagus, 2005), try to maximize the probability of sequences assuming the tokens are generated independently of each other.

3. Background

In this section, we describe how language model based sequence segmentation can be conceptualized via graphs in three parts: (1) constructing a graph where all possible segmentations (i.e., solutions) correspond with paths from a *source* to a *sink* node, (2) equating the shortest path search to the maximum likelihood model, (3) and extending the graph to reflect higher order probability models.

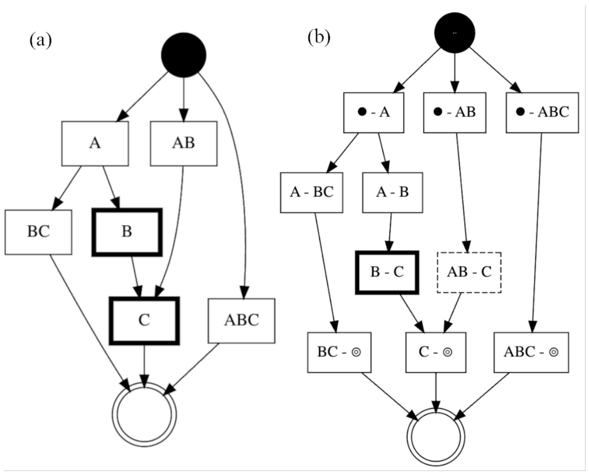

Figure 1: Illustrated unigram (a) and bigram (b) segmentation graphs for the example sequence $\mathtt{ABC}$. In bold, we indicate how an example pair of neighboring nodes, namely $\mathtt{B}$ and $\mathtt{C}$, corresponds with a single node in the bigram graph, $\mathtt{B}$–$\mathtt{C}$. Notice how, in the bigram graph, the production of the subsequence $\mathtt{C}$ is scored separately in the context of $\mathtt{B}$ and $\mathtt{AB}$ (dashed box).

3.1. From Sequences to Graphs

Suppose a sentence S of length m is the set of *atoms* a, each being a tuple $(\mathsf{position}, \mathsf{character})$:

$$\mathsf{S} = \{\mathsf{a}_1, ..., \mathsf{a}_m\} = \{(\mathsf{p}_1, \mathsf{c}_1), ..., (\mathsf{p}_n, \mathsf{c}_m)\}$$

To segment the sentence means to divide it into n subsets $\pi = \{\mathsf{w}_1, ..., \mathsf{w}_n\}$, which are (1) pairwise disjoint (non-overlapping), (2) exhaustive (spanning the entire sequence), and (3) subsequences, i.e., it must be possible to arrange the atoms of each w such that the difference between any two successive positions is equal to 1. We will denote by $\mathsf{Subseq}(\mathsf{S})$ the set of *all* candidate subsequences which can be formed from the original sentence.

Consider the example atomic sequence:

$$\mathsf{S} = \mathtt{ABC} = \{(1, \mathtt{A}), (2, \mathtt{B}), (3, \mathtt{C})\}.$$

To build the *unigram segmentation graph*, we first enumerate all $\mathsf{w} \in \mathsf{Subseq}(\mathsf{S})$, namely:

$$\mathsf{Subseq}(\mathsf{S}) = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{AB}, \mathtt{BC}, \mathtt{ABC}\}.$$

These form the basis for the graph vertices V. We create the edges E by connecting each vertex w_i to an other vertex w_j, if they are adjacent in S, i.e. the maximal atom position in w_i is exactly one less than the minimal position in w_j. The graph is then completed by including a special *begin* and *end*

	Word EN	TR	ZH	*Morfessor* EN	TR	ZH	*BERT* EN	TR	ZH
# tokens	61.9M	41.0M	51.6M	69.2M	58.4M	67.4M	64.9M	59.1M	79.7M
# types	262.8K	774.9K	476.4K	33.6K	180.6K	37.3K	24.7K	20.7K	10.5K
tokens/sent.	5.6	4.0	5.1	6.3	5.8	6.6	5.9	5.8	7.9
char./token	3.8	5.7	1.6	3.4	4.0	1.2	3.6	3.9	1.0

Table 1: Corpus statistics. Notice that, using the Huggingface BERT tokenizer, (1) the segmentation was nearly equivalent to character segmentation in case of Chinese (ZH; char./token=1), (2) on average, the tokens were almost one third shorter than words in Turkish (TR), (3) and, in English (EN), the tokens were only marginally shorter than words on average. A similar analysis holds for the case of Morfessor based segmentations, the main difference to the BERT segmentations being that Turkish and Chinese tokens were slightly longer, while English tokens were shorter, on average. The subword vocabularies were roughly an order of magnitude more compact than word vocabularies, the largest ones emerging in case of Turkish.

vertex, v_B and v_E, and (1) connecting the former to all vertices where the minimal position is 1 (thus $\mathtt{A}$, $\mathtt{AB}$, $\mathtt{ABC}$), and (2) connecting vertices where the maximal position is $|S| = 3$ (thus $\mathtt{C}$, $\mathtt{BC}$, $\mathtt{ABC}$) to v_E, the end vertex. See Figure 1a for an illustration.

Solving the segmentation problem now corresponds with finding the best path from v_B to v_E among the set of all such paths – the solution set – which we denote $\mathsf{Paths}(V, E)$.

3.2. Shortest Path and Maximum Likelihood

The data structure is now suitable for the decoding of the most likely sequence of segments according to a probabilistic language model. Interpreting the subsequences associated with the graph nodes as the outcomes of a categorical random variable, which is identically distributed (but not necessarily independent across position), we can assign each edge a weight that is based on the generation probability of the node it points to.

Scoring any particular segmentation, i.e. path $\pi \in \mathsf{Paths}(G)$, thus translates to computing the product of edge weights:

$$L(\pi) = \prod_{w \in \pi} p(w). \tag{1}$$

In practice, we maximize likelihood by minimizing its negative logarithm (NLL):

$$\pi_{\text{best}} = \underset{\pi \in \mathsf{Paths}(V,E)}{\arg\min} \; -\log(L(\pi)) \tag{2}$$

allowing us to score a candidate path as the sum of log-probabilities, because of this equivalence:

$$\log(\prod_{w \in \pi} p(w)) = \sum_{w \in \pi} \log(p(w)).$$

The problem formulation in terms of NLL is convenient, because conventional pathfinding algorithms are designed with the objective of minimizing the sum of edge weights.

3.3. Higher-Order Graphs

One way to create a higher order graph is by recursively creating a linegraph-like version of its previous-order graph, starting from the unigram version (similarly to how higher-order state-spaces are created in Markov models). Doing so once transforms each pair of adjacent nodes into a new node, now representing a bigram. An illustration of such a bigram graph can be seen in Figure 1b. The final shortest path in such a graph corresponds with a probability model involving a single additional dependency at every position in the sequence:

$$L(\pi) = p(w_1|v_B)...p(w_m|w_{m-1})p(v_E|w_m) \tag{3}$$

where the special vertices v_B and v_E can be interpreted as beginning-, and end-of-sentence tokens. By repeating the procedure, any n-gram graph can be derived.

4. Corpus

In the next sections, we present two experiments, both of which were conducted on the basis of OpenSubtitles [2] (Lison and Tiedemann, 2016), a movie subtitle corpus, which is part of the OPUS corpus (Tiedemann, 2012). We adapted the corpus by taking the overlapping set of documents (movies) between the English, Turkish and (simplified) Chinese subtitles. The resulting intersection was then pre-processed with the objective of keeping the alphabet minimal and language specific: (1) lowercasing, (2) removing punctuation, (3) removing characters that are not from the processed language, (4) and replacing all digit strings with the hashkey (#) character.

Lastly, we divided the corpus into a training and testing portion, using 90% of the subtitle lines for the former and 10% for the latter.

See the corpus statistics in Table 1.

[2] http://www.opensubtitles.org/

5. Experiment 1

The first experiment compares language modeling performance of three (sub-)word segmentation strategies as a function of context size.

5.1. Segmentation Strategies

We selected three competing types of segmentation: word segmentation, Morfessor based subword segmentation, and segmentation produced by popular tokenizers from the Huggingface python library [3].

Word segmentation was obtained by simply taking the tokenized versions of the subtitle corpus. According to Lison and Tiedemann (2016), the English subtitles were tokenized by the Moses toolkit (Koehn et al., 2007) and the Chinese subtitles were tokenized by the KyTea library (Neubig et al., 2011).

Morphologically similar *subword* segmentation was obtained via the Morfessor library (Virpioja et al., 2013). For English and Turkish, we trained the unsupervised baseline model on word counts provided by the latest edition of the MorphoChallenge (Kurimo et al., 2010); for Chinese, we trained on word counts derived from the training split of the subtitle corpus.

BERT tokenizer *subword* segmentations were obtained from the following pre-trained Huggingface models: 'bert-base-uncased' for English, 'dbmdz/bert-base-turkish-uncased' for Turkish, and 'bert-base-chinese' for Chinese. These tokenizers are variants of the BPE (Sennrich et al., 2016) algorithm and are arguably the most widely used text segmenters in industry and academic research related to modern language models.

5.2. Evaluation

For every combination of language and segmentation type, we fitted an *n-gram* count-based language model of order up to 5 on the training split of the dataset, and evaluated it on the testing split.

The results are reported as average values of *bits-per-sentence* (BPS), i.e. the sum of negative log-probabilities of tokens in one line of the test corpus:

$$\mathrm{BPS}(\pi) = -\log(\mathrm{L}(\pi))$$

where $\mathrm{L}(\pi)$ is defined by Equation 1 for unigram models, and Equation 3 for n-gram models where $\mathrm{n} > 1$. We used BPS rather than BPC[4] (bits-per-character), because the former allows for easier comparison between languages and, in this case, only skews the results negligibly since the information content was roughly controlled for by using the same set of movies for each language.

Merely reporting the n-gram order as context size would be misleading, because the size of a particular n-gram with respect to the sentence size depends on the segmentation strategy and language. To account for this variability, we report the context size as the fraction of sentence length (in characters) that the portion of the n-gram used as context amounts to on average. The exact value was computed according to the following formula:

$$(\mathrm{n}-1) * \overline{\mathrm{TL}} / (\overline{\mathrm{SL}} + \mathrm{n} - 1)$$

where n denotes the order of the n-gram, and $\overline{\mathrm{TL}}$, $\overline{\mathrm{SL}}$ denote the mean token and sentence length in characters. The term $\mathrm{n}-1$ is added to the mean sentence length in the denominator because we pre-pended one single-character padding token for every n-gram order increase above 1 to every sentence.

We used the NLTK (Bird et al., 2009) implementation and the back-off strategy (Katz, 1987) to score unseen words and n-grams: if a particular n-gram does not exist, an $(\mathrm{n}-1)$-gram (containing one less context token) is attempted. If all of the attempts – including the unigram – fail, a logscore derived from frequency 1 is used.

5.3. Results

The results are visualized in Figure 2.

A shared pattern across settings is the reduction of surprisal with higher amount of context.

In the case of English, the differences in scores between the segmentation methods were very small across the entire observed range of context size, and, in contrast to Turkish and Chinese, the word segmentation scored marginally but overall better than the other segmentations.

With Turkish, there was a stronger difference between segmentation methods in terms of surprisal reduction rate. While the word segmentation was still the most optimal at the unigram setting, at around 20% of sentence length used as context, the ranking reversed in favor of the segmentations with shorter tokens: Morfessor and BERT. Between these two, however, the difference was minimal to none.

For Chinese, we observed a similar pattern of ranking reversal at around 20% of context size. The difference to the case of Turkish was mainly an overall faster decline of surprisal values, and an additional difference in rates between Morfessor and BERT segmentations, the latter ranking as most optimal and fastest declining.

[3]https://huggingface.co

[4]$\mathrm{BPC} = \mathrm{BPS}/|\mathrm{S}|$

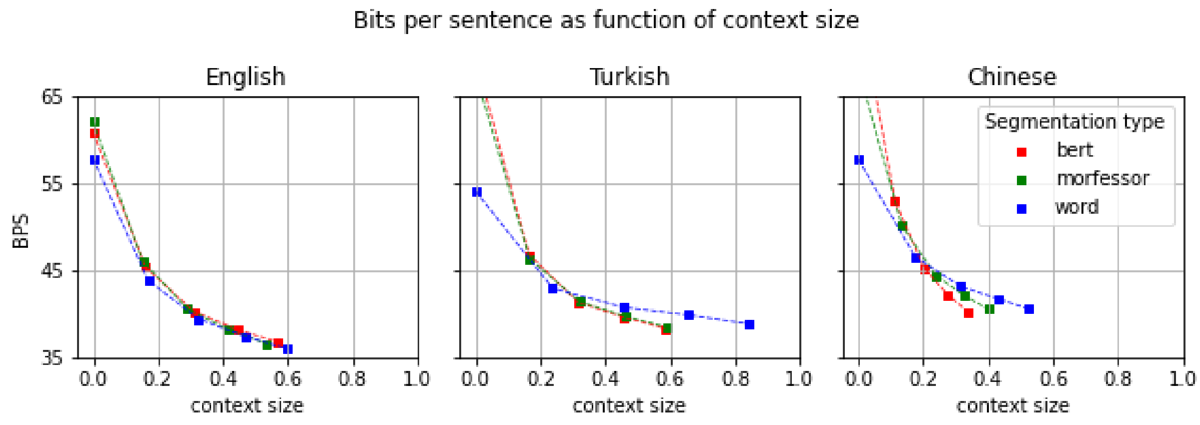

Figure 2: Results of the Experiment 1. The y-axis represents bits-per-sentence on the same scale for each language. The x-axis values correspond with context size measured as proportion of sentence length, 0 implying the unigram language model. The change in model order that modulates this metric (1 to 5, in increments of 1) results in different sentence length proportions due to differently sized tokens: a single Turkish token covers, on average, longer fractions of the sentence than an English or a Chinese token.

5.4. Discussion

Our results indicate that the optimal elementary unit of analysis for language modeling is not universal, but that it depends on the specific characteristics of the language and writing systems.

Previous experiments in language modeling with Chinese text demonstrated better performance for character-based models compared to their word-based counterparts (Li et al., 2019; Mielke et al., 2019), which aligns with our result of word segmentation having higher surprisal rates than character-based segmentation (BERT). Then again, the slightly coarser Morfessor tokens did better than character-based segmentation, indicating that some chunking of Chinese characters might be meaningful.

Similarly, in a study about the impact of Turkish tokenization on language model performance, Toraman et al. (2023) reported that models trained on finer-grained BPE-based segmentations outperform more coarse morphological, word, and character based segmentations. Similarly, the BERT and Morfessor-based segmentation outperformed the word-based segmentation in our experiments.

Previous work comparing English tokenization strategies mainly focused on subword segmentations and recommends using morphologically aligned segmentations over BPE-based techniques (Mielke et al., 2019; Bostrom and Durrett, 2020; Park et al., 2021). However, our results suggest that the English word is not less optimal than other subword units for segmentation. This may have been overlooked in other studies in which the size of context used to predict the next token was not properly controlled for. Word-based segmentation may also be less useful in practice because it requires a larger vocabulary.

Our finding that subword segmentations in Turkish and Chinese benefit from more context is somewhat puzzling, since both Morfessor and BPE discard between-token dependencies during training. In the case of Morfessor, the addition of a prior term regulating vocabulary size in the training procedure could be a contributing factor.

6. Experiment 2

In the second experiment, the goal was to assess the benefits of using a higher-order language model to detect *word* boundaries. To put the effect in perspective, we compare it to a baseline effect of using an increasingly large language sample as the basis for the language model.

To simulate word segmentation, we deleted any within-sentence word boundary notation (i.e. all punctuation and whitespaces) from the test corpus on a per-sentence basis. Every such sentence was then segmented by, first, constructing a segmentation graph (as described in Sections 3.1 and 3.3) and, second, finding the shortest path (defined by Equation 2). The procedure was repeated using uni-, bi-, and trigram language models fitted on word counts derived from increasingly larger samples of sentences, i.e. movie subtitle lines, of the training corpus. The sentences were sampled without replacement and in quantities ranging from 10^2 to 10^7 with integer increments in order of magnitude. The 7th order of magnitude was the full corpus; for all orders of magnitude less than that, we collected 3 differently seeded samples to account for random variation.

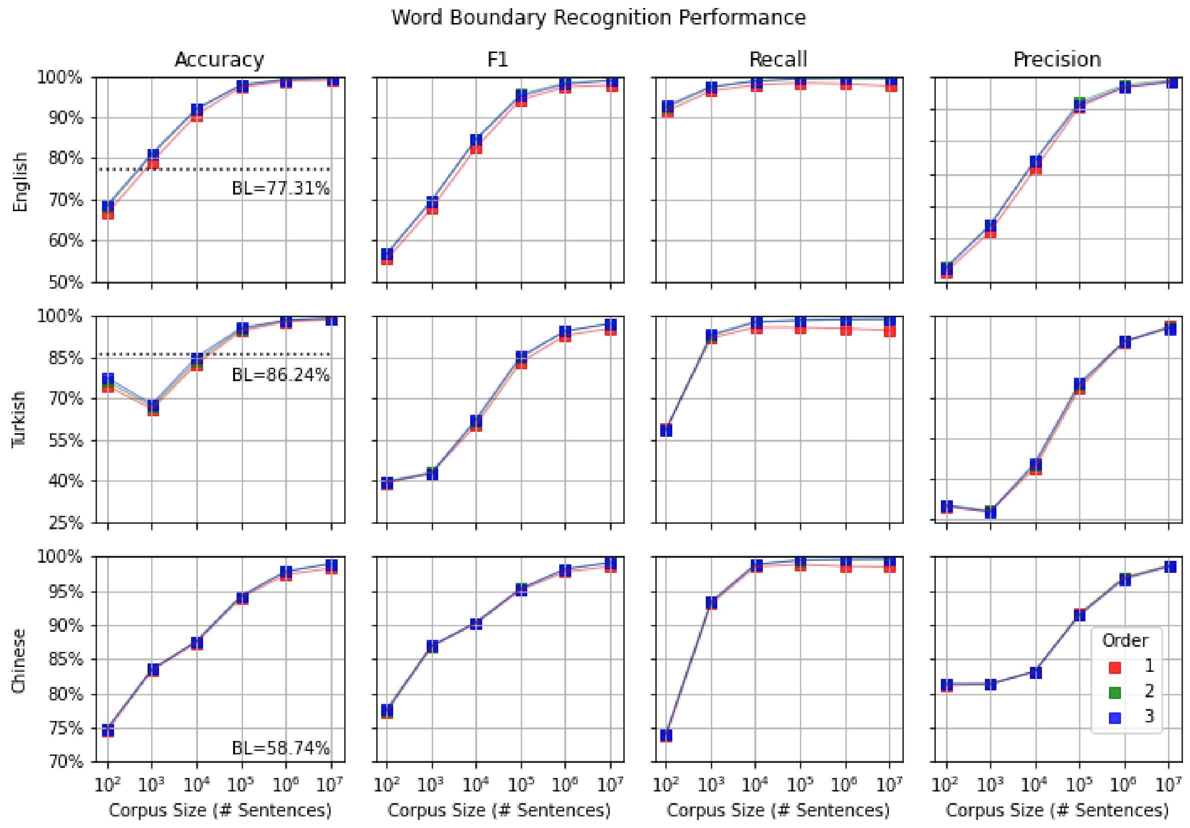

Figure 3: Illustrated results of Experiment 2. The x-axis depicts the corpus size as number of sentences on a log-10 scale. Notice that, while the top value on the y-axis is always 100%, the bottom value is language-specific due to widely different ranges overall. The points in green, corresponding with bigram segmentation models, mostly overlap with the blue (trigram) points.

6.1. Evaluation

The solutions – paths in the segmentation graphs – were converted to sequences of positive and negative labels that correspond with potential word boundaries and indicate whether the position is a boundary or not. The task was evaluated as binary classification, using accuracy, F1-score, recall and precision as performance metrics. The labels were not well balanced: more negative than positive cases are to be expected, but to a different extent for each language. The proportion of cases with the majority label determines our baseline accuracy values.

We evaluated the results on a heldout test-set containing 10000 sentences. The test set differed slightly for each setting of the language model n-gram order: for the unigram model, the sentences were up to 100 characters long, for the bigram model up to 40, and for the trigram model up to 30. We did this because of the different computational demands of the higher-order segmentation. In practice, this means that the unigram models were evaluated on a larger test-set, although only by a little, because the distribution of sentence length was skewed towards values within the 30 character range.

6.2. Results

The results are listed in Table 2. Given the overall trends in Figure 3, we decided to aggregate the order effects separately for small and large models, corresponding with the left and right ranges of corpus size in the figure's plots.

One immediate observation in the Figure 3 is that, in comparison to the baseline effect of sample size, the model order had minimal effect on all performance metrics. The largest, among the small model order effects, was that of bigram vs. unigram model i.t.o. recall. The increment to trigram model, mostly resulted in no further benefits. Overall, the accuracy values closely approached the 100% mark in all three languages when the sample size was the largest. Another general trend was that of rapid recall onset, but lagging rise in precision, which also manifested in F1-scores somewhat lagging behind accuracy scores. With recall, although being high in general, there was a subtle decreasing trend in case of all unigram models.

	English	Turkish	Chinese
Order Effect (models sized 10^2 to 10^4)			
1 to 2	1.53 ± .45	1.22 ± .96	0.21 ± .11
2 to 3	0.31 ± .22	1.15 ± 1.02	0.00 ± .02
Order Effect (models sized 10^5 to 10^7)			
1 to 2	0.53 ± .08	0.54 ± .10	0.41 ± .17
2 to 3	-0.11 ± .06	0.23 ± .19	-0.05 ± .03
Sample Size Effect			
2 to 3	12.55 ± 1.42	-9.06 ± 2.19	8.73 ± 0.78
3 to 4	11.14 ± 0.79	16.70 ± 1.33	4.07 ± 0.53
4 to 5	6.15 ± 0.47	11.48 ± 0.81	6.61 ± 0.55
5 to 6	1.39 ± 0.09	3.15 ± 0.24	3.51 ± 0.14
6 to 7	0.35 ± 0.08	0.77 ± 0.04	1.01 ± 0.14

Table 2: Results of the Experiment 2. The unit value is percentage of accuracy in word boundary classification. The upper portion of the table aggregates the effects of increasing the model order in language models trained on small corpora, the center part on larger corpora. The lower part lists the effects of corpus size i.t.o. increments in order of magnitude.

In English, the increase in accuracy was largely due to the sample size increments from 10^2 to 10^5. From 10^6 on, the accuracy was $> 99\%$. The effect of increasing the model order from 1 to 2 in the smaller models was dwarfed by the sample size effect; with larger models, however, its value of 0.53% was non-negligible compared to the sample size effects $(1.39\%, 0.35\%)$.

In Turkish, we found the singular case of decrease in accuracy due to an increase in sample size, namely from 10^2 to 10^3, matching a dip in precision at this value. The baseline accuracy for Turkish was the highest, and surpassed only by models trained on 10^5 and more sentences. The observations about the effect of model order in English also translate to Turkish.

In Chinese, although the accuracy values grew the slowest, the difference to baseline values was the largest. The pattern of quick onset of recall and lagging precision was also the most marked. The effect of model order was weak with the smaller models, but, with larger models, the increment from 1 to 2 resulted in an accuracy increase of 0.41%, which is non-negligible in comparison to the sample size effect from 10^6 to 10^7 of 1.01%.

6.3. Discussion

The results indicate that, for statistical word segmentation, working with a high quality language sample is important. Segmenting the text with a bigram instead of unigram model can result in further increase in accuracy, although this effect is subtle and only relevant once the language sample is representative enough.

This finding supports the current trend of using unigram-decoder based text tokenizers, which are convenient for their low computational requirements. However, for use-cases where accuracy matters, such as recovering words or morphemes – tokens with precise linguistic definitions –, bigram model based segmentation is recommended. In future work, it would be interesting to explore whether higher-order segmentation aids in, e.g., morphological segmentation or syllabification.

The decline in recall between the unigram and bigram based segmentations is in line with the findings of Goldwater et al. (2009), who connected the independence assumption to undersegmentation. In our findings, larger unigram models did not have problems over-diagnosing boundaries. Although the sensitivity dropped somewhat for unigram models, the higher-order models did not suffer a decline.

7. General Discussion

The two presented experiments explore two different aspects of the role of context in text segmentation. The first experiment examined the difference context makes when evaluating competing segmentation methods. The second experiment looked at the effect of context on statistical word segmentation.

The results suggest that context plays a definitive role in evaluating segmentation methods: the optimal way to encode language is specific to the amount of context used for discovering the regularities in token occurrence. However, we observed, that this is language specific. Surprisingly, our findings also revealed that English word segmentation was on par with the two subword segmentations.

Looking at statistical word segmentation only, the role of context was observable but in small magnitude. While perhaps trivial, this observation is reassuring. It suggests that the inference of distributions governing the sub-lexical regularities (i.e., tokenizers) does not depend on jointly inferring super-lexical regularities, which would severely complicate the procedure. It further implies that, to the extent that written text mirrors properties of spoken language, this offers an explanation on how children can learn to discern words while being unaware of higher-level dependencies between them due to, e.g., syntax or semantics, which they learn at later stages of development.

8. Conclusion

The role of preceding text in tokenization was manifested in two ways. When comparing the difficulty in modeling differently tokenized corpora, our results indicate that the assessment may fully reverse when context is involved compared to when it is absent. In light of a word segmentation experiment, the role was more subtle: word boundaries were only marginally more accurately recognized when using context-sensitive, rather than context free, methods to score the hypotheses.

9. Acknowledgements

This research was funded by ITK as part of the SmartDATA (10028312) project as part of the KPN Responsible AI lab. We also thank the reviewers for their insightful comments.

10. Limitations

In this work, we only computed language modeling surprisal values on the basis of count-based language models. The extent to which these results generalize to other types of language models (e.g. neural network based) is unclear.

11. Bibliographical References

Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, Jie Chen, Jingdong Chen, Zhijie Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Ke Ding, Niandong Du, Erich Elsen, Jesse Engel, Weiwei Fang, Linxi Fan, Christopher Fougner, Liang Gao, Caixia Gong, Awni Hannun, Tony Han, Lappi Johannes, Bing Jiang, Cai Ju, Billy Jun, Patrick LeGresley, Libby Lin, Junjie Liu, Yang Liu, Weigao Li, Xiangang Li, Dongpeng Ma, Sharan Narang, Andrew Ng, Sherjil Ozair, Yiping Peng, Ryan Prenger, Sheng Qian, Zongfeng Quan, Jonathan Raiman, Vinay Rao, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Kavya Srinet, Anuroop Sriram, Haiyuan Tang, Liliang Tang, Chong Wang, Jidong Wang, Kaifu Wang, Yi Wang, Zhijian Wang, Zhiqian Wang, Shuang Wu, Likai Wei, Bo Xiao, Wen Xie, Yan Xie, Dani Yogatama, Bin Yuan, Jun Zhan, and Zhenyao Zhu. 2016. Deep speech 2 : End-to-end speech recognition in english and mandarin. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 173–182, New York, New York, USA. PMLR.

Rie Kubota Ando and Tong Zhang. 2005. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853.

M A Anusuya and S K Katti. 2009. Speech recognition by machine: A review. *IJCSIS) International Journal of Computer Science and Information Security*, 6.

Eleanor Olds Batchelder. 2002. Bootstrapping the lexicon: A computational model of infant speech segmentation. *Cognition*, 83:167–206.

Yves Bestgen. 2006. Improving text segmentation using latent semantic analysis: A reanalysis of choi, wiemer-hastings, and moore (2001). *Computational Linguistics*, 32:5–12.

Narin Bi and Nguonly Taing. 2014a. Khmer word segmentation based on bi-directional maximal matching for plaintext and microsoft word document. In *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2014 Asia-Pacific*, pages 1–9. IEEE.

Narin Bi and Nguonly Taing. 2014b. Khmer word segmentation based on bi-directional maximal matching for plaintext and microsoft word document. In *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2014 Asia-Pacific*, pages 1–9.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Kaj Bostrom and Greg Durrett. 2020. Byte pair encoding is suboptimal for language model pretraining. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4617–4624, Online. Association for Computational Linguistics.

Michael R Brent. 1999. Speech segmentation and word discovery: A computational perspective. *Trends in Cognitive Sciences*, 3(8):294–301.

Vichet Chea, Ye Kyaw Thu, Chenchen Ding, Masao Utiyama, Andrew Finch, and Eiichiro Sumita. 2015. Khmer word segmentation using conditional random fields. *Khmer Natural Language Processing*, pages 62–69.

Aitao Chen. 2003. Chinese word segmentation using minimal linguistic knowledge. In *Proceedings of the Second SIGHAN Workshop on Chinese Language Processing*, pages 148–151, Sapporo, Japan. Association for Computational Linguistics.

Aitao Chen, Jianzhang He, Liangjie Xu, Fredric C. Gey, and Jason Meggs. 1997. Chinese text retrieval without using a dictionary. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '97, page 42–49, New York, NY, USA. Association for Computing Machinery.

Keh-Jiann Chen and Shing-Huan Liu. 1992. Word identification for mandarin chinese sentences.

Grzegorz Chrupała. 2023. Putting natural in natural language processing. *arXiv preprint arXiv:2305.04572*.

Ronald A Cole, Jola Jakimik, William E Cooper, and Joia Jakimik. 1980. Segmenting speech into words. *J. Acoust. Soc. Am*, 67:1323–1332.

Mathias Creutz and Krista Lagus. 2005. *Unsupervised morpheme segmentation and morphology induction from text corpora using Morfessor 1.0*. Helsinki University of Technology Helsinki.

Mathias Creutz and Krista Lagus. 2007. Unsupervised models for morpheme segmentation and morphology learning. *ACM Transactions on Speech and Language Processing*, 4.

Matthew H. Davis, William D. Marslen-Wilson, and M. Gareth Gaskell. 2002. Leading up the lexical garden path: Segmentation and ambiguity in spoken word recognition. *Journal of Experimental Psychology: Human Perception and Performance*, 28:218–244.

Ke Deng, Peter K. Bol, Kate J. Li, and Jun S. Liu. 2016. On the unsupervised analysis of domain-specific chinese texts. *Proceedings of the National Academy of Sciences of the United States of America*, 113:6154–6159.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Miguel Domingo, Mercedes Garcıa-Martınez, Alexandre Helle, Francisco Casacuberta, and Manuel Herranz. 2019. How much does tokenization affect neural machine translation?

Nadir Durrani and Sarmad Hussain. 2010. Urdu word segmentation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 528–536.

Schubert Foo and Hui Li. 2004. Chinese word segmentation and its effect on information retrieval. *Information Processing and Management*, 40(1):161–190.

Philip Gage. 1994. A new algorithm for data compression. *C Users J.*, 12(2):23–38.

Matthias Gallé. 2019. Investigating the effectiveness of BPE: The power of shorter sequences. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1375–1381, Hong Kong, China. Association for Computational Linguistics.

Xianping Ge, Wanda Pratt, and Padhraic Smyth. 1999. Discovering chinese words from unsegmented text. *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1999*, pages 271–272.

Sharon Goldwater, Thomas L. Griffiths, and Mark Johnson. 2009. A bayesian framework for word segmentation: Exploring the effects of context. *Cognition*, 112:21–54.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring network structure, dynamics, and function using networkx. In *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA.

Awni Y. Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. 2014. Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567.

Matthias Huck, Simon Riess, and Alexander Fraser. 2017. Target-side word segmentation strategies for neural machine translation. In *Proceedings of the Second Conference on Machine Translation*, pages 56–67, Copenhagen, Denmark. Association for Computational Linguistics.

Chea Sok Huor, Top Rithy, Ros Pich Hemy, Vann Navy, Chin Chanthirith, and Chhoeun Tola. 2004a. Word bigram vs orthographic syllable bigram in khmer word segmentation. *PAN Localization Working Papers*, 2007.

Chea Sok Huor, Top Rithy, Ros Pich Hemy, Vann Navy, Chin Chanthirith, and Chhoeun Tola. 2004b. Word bigram vs orthographic syllable bigram in khmer word segmentation. *PAN Localization Working Papers*, 2007.

Peter W. Jusczyk, Anne Cutler, and Nancy J. Redanz. 1993. Infants' preference for the predominant stress patterns of english words. *Child Development*, 64:675.

S. Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401.

Dennis H. Klatt and Kenneth N. Stevens. 1973. On the automatic recognition of continuous speech: Implications from a spectrogram-reading experiment. *IEEE Transactions on Audio and Electroacoustics*, 21:210–217.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic. Association for Computational Linguistics.

Hideki Kozima. 1996. Text segmentation based on similarity between words.

Taku Kudo. 2006. Mecab: Yet another part-of-speech and morphological analyzer. `https:`

//taku910.github.io/mecab/. Accessed: 2023-12-04.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing.

Mikko Kurimo, Sami Virpioja, Ville T. Turunen, Graeme W. Blackwood, and William Byrne. 2010. Overview and results of Morpho Challenge 2009. In *Multilingual Information Access Evaluation I. Text Retrieval Experiments: 10th Workshop of the Cross-Language Evaluation Forum, CLEF 2009, Corfu, Greece, September 30 – October 2, 2009, Revised Selected Papers*, volume 6241 of *Lecture Notes in Computer Science*, pages 578–597. Springer Berlin / Heidelberg.

Mun-Kew Leong and Hong Zhou. 1997. Preliminary qualitative analysis of segmented vs bigram indexing in chinese.

Xiaoya Li, Yuxian Meng, Xiaofei Sun, Qinghong Han, Arianna Yuan, and Jiwei Li. 2019. Is word segmentation necessary for deep learning of chinese representations?

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

Józef Maciuszek. 2018. Lexical access in the processing of word boundary ambiguity. *Social Psychological Bulletin*, 13.

Yuji Matsumoto, Akira Kitauchi, Tatsuo Yamashita, Yoshitaka Hirano, Hiroshi Matsuda, Kazuma Takaoka, and Masayuki Asahara. 2000. Morphological analysis system chasen version 2.2. 1 manual. *Nara Institute of Science and Technology*.

Sven L. Mattys, Peter W. Jusczyk, Paul A. Luce, and James L. Morgan. 1999. Phonotactic and prosodic effects on word segmentation in infants. *Cognitive Psychology*, 38:465–494.

Surapant Meknavin, Paisarn Charoenpornsawat, and Boonserm Kijsirikul. 1997. Feature-based thai word segmentation.

S Meknawin. 1995. Towards 99.99% accuracy of thai word segmentation. In *Oral Presentation at the Symposium on Natural Language Processing in Thailand*, volume 95.

Sabrina J. Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y. Lee, Benoît Sagot, and Samson Tan. 2021. Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp.

Sabrina J. Mielke, Ryan Cotterell, Kyle Gorman, Brian Roark, and Jason Eisner. 2019. What kind of language is hard to language-model? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4975–4989, Florence, Italy. Association for Computational Linguistics.

Tomáš Mikolov, Ilya Sutskever, Anoop Deoras, Hai-Son Le, Stefan Kombrink, and Jan Cernocky. 2012. Subword language modeling with neural networks. *preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf)*, 8(67).

Graham Neubig, Yosuke Nakata, and Shinsuke Mori. 2011. Pointwise prediction for robust, adaptable Japanese morphological analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 529–533, Portland, Oregon, USA. Association for Computational Linguistics.

Jian Yun Nie, Martin Brisebois, and Xiaobo Ren. 1996. On chinese text retrieval. *Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 225–234.

Shu Okabe, Laurent Besacier, and François Yvon. 2022. Weakly supervised word segmentation for computational language documentation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7385–7398, Dublin, Ireland. Association for Computational Linguistics.

Hyunji Hayley Park, Katherine J Zhang, Coleman Haley, Kenneth Steimel, Han Liu, and Lane Schwartz. 2021. Morphology matters: a multilingual language modeling analysis. *Transactions of the Association for Computational Linguistics*, 9:261–276.

Lawrence Phillips and Lisa Pearl. 2014. Bayesian inference as a cross-linguistic word segmentation strategy: Always learning useful things. In *Proceedings of the 5th Workshop on Cognitive*

Aspects of Computational Language Learning (CogACLL), pages 9–13, Gothenburg, Sweden. Association for Computational Linguistics.

Yuen Poowarawan. 1986. Dictionary-based thai syllable separation. In *Proc. Ninth Electronics Engineering Conference (EECON-86), Thailand*, pages 409–418.

A Pornprasertkul. 1994. *Thai syntactic analysis*. Ph.D. thesis, Ph. D Thesis, Asian Institute of Technology.

S Rarunrom. 1991. Dictionary-based thai word separation. *Senior Project Report*.

D. Raj Reddy. 1976. Speech recognition by machine: A review. *Proceedings of the IEEE*, 64:501–531.

Jenny R Saffran, Richard N Aslin, and Elissa L Newport. 1996a. Statistical learning by 8-month-old infants. *New Series*, 274:1926–1928.

Jenny R. Saffran, Elissa L. Newport, and Richard N. Aslin. 1996b. Word segmentation: The role of distributional cues. *Journal of Memory and Language*, 35:606–621.

Masahiko Sato. 1999. Kakasi - kanji kana simple inverter. `http://kakasi.namazu.org`. Accessed: 2023-12-04.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Richard Sproat and Thomas Emerson. 2003. The first international Chinese word segmentation bakeoff. In *Proceedings of the Second SIGHAN Workshop on Chinese Language Processing*, pages 133–143, Sapporo, Japan. Association for Computational Linguistics.

Richard Sproat and Chilin Shih. 1990. A statistical method for finding word boundaries in chinese text. *Computer Processing of Chinese and Oriental Languages*, 4:336–351.

Maosong Sun, Dayang Shen, and Benjamin K. Tsou. 1998. Chinese word segmentation without using lexicon and hand-crafted training data. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 2*, pages 1265–1271, Montreal, Quebec, Canada. Association for Computational Linguistics.

Zhiqing Sun and Zhi Hong Deng. 2018. Unsupervised neural word segmentation for chinese via segmental language modeling. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018*, pages 4915–4920.

Cagri Toraman, Eyup Halit Yilmaz, Furkan Şahinuç, and Oguzhan Ozcelik. 2023. Impact of tokenization on language models: An analysis for turkish. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, 22(4).

Channa Van and Wataru Kameyama. 2013. Khmer word segmentation and out-of-vocabulary words detection using collocation measurement of repeated characters subsequences. *GITS/GITI Research Bulletin*, 2012-2013:21–31.

Anand Venkataraman. 2001. A statistical model for word discovery in transcribed speech. *Computational Linguistics*, 27:351–372.

Sornlertlamvanich Virach. 1993. Word segmentation for thai in machine translation system. *Machine translation*.

Sami Virpioja, Peter Smit, Stig-Arne Grönroos, Mikko Kurimo, et al. 2013. Morfessor 2.0: Python implementation and extensions for morfessor baseline. Technical report, Aalto University.

Hai Zhao, Deng Cai, Changning Huang, and Chunyu Kit. 2019. Chinese word segmentation: Another decade review (2007-2017).

Hai Zhao and Chunyu Kit. 2008. An empirical comparison of goodness measures for unsupervised Chinese word segmentation with a unified framework. In *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-I*.

12. Language Resource References

Lison, Pierre and Tiedemann, Jörg. 2016. *Opensubtitles2016: Extracting large parallel corpora from movie and tv subtitles*. European Language Resources Association (ELRA).

Tiedemann, Jörg. 2012. *Parallel Data, Tools and Interfaces in OPUS*. European Language Resources Association (ELRA).

Abbreviation across the world’s languages and scripts

Kyle Gorman & Brian Roark
Google Research
kbg@google.com, roark@google.com

Abstract

Detailed taxonomies for *non-standard words*, including abbreviations, have been developed for speech and language processing, though mostly with reference to English. In this paper, we examine abbreviation formation strategies in a diverse sample of more than 50 languages, dialects and scripts. The resulting taxonomy—and data about which strategies are attested in which languages—provides key information needed to create multilingual systems for abbreviation expansion, an essential component for speech processing and text understanding.

Keywords: abbreviation, abbreviation expansion, text normalization, writing systems

1. Introduction

One of the oldest and most entrenched features of written language, dating back to the dawn of history, is the use of abbreviatory devices. Gorman et al. (2021) point out that nearly a third of the Latin words inscribed at the base of Trajan’s Column in the Roman forum—completed early in the second century CE—are abbreviations, such as ⟨TRIB POT⟩ for *tribunicia potestate* ‘power of the tribune’. Though abbreviation seems to be general feature of written language, most discussion of this property in the speech and language processing community has focused on English, and to a lesser degree other languages written with the Latin alphabet. Much less work has focused on abbreviation formation in other languages or writing systems. In this work we take first steps towards a more inclusive documentation of abbreviation in the world’s languages and scripts.

Detailed—albeit dated—treatments of English abbreviation formation are provided by Marchand (1969, §9) and Cannon (1989). As Cannon’s extensive bibliography attests, there is a long and robust literature documenting English acronyms.

More recently, speech and language processing specialists have developed data-driven *abbreviation expansion* engines for converting abbreviations to fully-expanded words, usually using nearby linguistic context to resolve ambiguities (e.g., Baldwin et al., 2015; Chrupała, 2014; Roark and Sproat, 2014; Gorman et al., 2021). Unfortunately, this literature is also dominated by work solely considering English.[1] Of course, key to these sorts of abbreviation expansion methods is the availability of data, and unfortunately there is little of that, even for the best resourced languages. Several of the above cited works make use of ad hoc methods to detect or synthesize abbreviations in context (Roark and Sproat, 2014; Żelasko, 2018), whereas others rely on social media data from shared tasks (e.g., Baldwin et al., 2015; Chrupała, 2014), data that is unfortunately no longer available. To the authors’ knowledge, the data set from Gorman et al. (2021),[2] produced by asking annotators to shorten selected sentences from English-language Wikipedia, is the only large-scale abbreviation data set available to the public.

One limitation of previous computational work on abbreviation expansion—reflecting its narrow focus on a relatively small number of languages and scripts—is the simplifying assumption that all abbreviations are formed by deleting one or more characters (i.e., they are *deletion-based* in the sense of Pennell and Liu 2010). Formally, then, this assumes that abbreviations are proper subsequences of their corresponding full forms, with no further augmentations or changes to spelling. Furthermore, some of this prior work also focuses on abbreviations of a single word and ignores abbreviations formed from phrases. As will be seen below, neither of these assumptions is generally valid for the world’s languages and scripts.

In this study, we describe the collection of a survey of abbreviatory mechanisms used in a diverse sample of just over 50 languages, dialects, and scripts. From the results of the survey, we provide a taxonomy of abbreviation expansion strategies, and provide information about which abbreviation strategies, if any, are attested in each of the surveyed languages, dialects, and scripts. We thus document the kinds of phenomena that can be found in this diverse sample. We conjecture that it will always be necessary to impose constraints on what abbreviatory strategies are considered, so this information is a prerequisite for building and validating the data collection and annotation processes needed to build truly multilingual abbrevia-

[1]One notable exception is Żelasko (2018), who studies the particular challenges of abbreviation expansion in Polish, a richly inflected language.

[2]`https://github.com/google-research-datasets/WikipediaAbbreviationData`

tion expansions systems. All survey data, including examples elicited from the language consultants, is publicly available under a Creative Commons CC-BY 4.0 license.[3]

2. Background

Abbreviations are a class of what Sproat et al. (2001) call *non-standard words*, forms found in written text which are generally not pronounced according to the ordinary letter-to-sound rules of the language. Non-standard words, henceforth NSWs, pose difficulties for speech and text processing applications. In particular, text-to-speech (TTS) systems require NSWs to be converted to "spoken form" (e.g., Ebden and Sproat, 2015), and automatic speech recognition (ASR) systems must invert this transduction so that the resulting transcripts can be displayed to users in a readable format (e.g., Pusateri et al., 2017).[4] Sproat et al. (2001) provide a taxonomy of NSWs, and this is further enriched by van Esch and Sproat (2017). These taxonomies provide useful information for the linguists and engineers who design the many computer systems that interact with NSWs. Sproat et al. propose three broad categories of abbreviation, focusing on the pronunciation of the NSW token: ASWDs: those read as a word (e.g., *NATO*); LSEQs: those read as a letter sequence (e.g., *CIA*); and EXPNs: general abbreviations (e.g., *Blvd.*). In this work, however, we propose a taxonomy that goes beyond mere pronunciation.

Gorman et al. (2021) discern two broad classes of abbreviations. First are *conventionalized* abbreviations, which are of high enough frequency that both their pronunciation and denotation are known to most readers, at least in certain speech communities or text genres. These include SI[5] units (e.g., *Hz* read as *Hertz*) and abbreviations for certain geographic entities (e.g., *OH* read as *Ohio*). The second type are *ad-hoc* abbreviations, those coined as needed. These are particularly common on digital communications channels which prefer brevity, such as text messaging (e.g., Crystal, 2001, 2008). These ad-hoc abbreviations are rarely recorded by lexicographers.

Like other NSWs, it is not always obvious how one might read an abbreviation. For instance, one might read *NATO*—a conventionalized abbreviation—as a word rhyming with *Cato*, or possibly expand it to its full form, the *North Atlantic Treaty Organization*, but it is not ordinarily read letter by letter. In contrast, *CIA* is an *initialism*, an abbreviation which is read letter by letter, but never as a two-syllable word rhyming with *Garcia*. Such language- and word-specific facts are crucial for building an ASR verbalizer or a TTS front-end.

Abbreviations, particularly ad-hoc abbreviations, pose an additional difficulty not common to other NSWs: it is not always obvious what they denote. For instance, *AMA* could represent: *American Medical Association*, a professional organization and lobbying group for doctors; *against medical advice*, jargon referring to a patient leaving the hospital before being cleared for discharge; or *ask me anything*, a prompt used on various online forum to solicit questions; and this does not exhaust the possibilities. Similarly, the ad-hoc abbreviation *brd* might denote *bread*, *broad*, or *bird*, and context is required to determine how to pronounce or interpret it. For this reason, methods for abbreviation expansion must take context into account.

3. Methods

We conducted a survey of abbreviatory methods used in diverse languages and scripts. Adapting pre-existing practices at Google for internationalizing text normalization systems, we hypothesized that literate, linguistically sophisticated language consultants would have reasonably reliable judgments about whether or not a particular type of abbreviation formation process is present in their language when provided with examples of that process. These examples were in English—the language used for the survey instrument itself—when relevant examples exist in English, and glosses were provided for examples from other languages when relevant examples do not exist in English. Table 1 lists the seven abbreviation strategies targeted and the examples provided; the full text of the survey can be found in Appendix A.

To strengthen conclusions about which types of abbreviations are present in which language, we also asked consultants to provide two or three examples of each process they claimed for their language.[6] Elicitation of examples allowed us to discern whether a consultant understood the strategies during the consensus procedure.

[3] `https://github.com/google-research/google-research/tree/master/multilingual_abbreviation_survey`

[4] Such transductions are traditionally referred to as *text normalization*, though this term has taken on a broader sense—since it was first popularized by Sproat et al., 2001—which includes the conversion of noisy user-generated text to conventional spelling for purposes unrelated to speech processing.

[5] 'International System of Units', whose abbreviation comes from the French *Système international d'unités*.

[6] We note in passing that requiring an additional step when answering "yes" might cause consultants to have a response bias in favor of "no". While the consensus procedure described in subsection 3.4 is intended to avoid errors due to this bias, we have no straightforward way to detect the presence of such a bias.

Abbreviation class	Example
First-character abbreviations	*NATO* ($<$ *North Atlantic Treaty Organization*)
Stump compounds	*FiDi* ($<$ *Financial District*)
Truncations	*Col.* ($<$ *Colonel*)
Augmented truncations	Australian English *footie* ($<$ *football* plus augment *-ie*)
Word-internal deletions	*Blvd.* ($<$ *Boulevard*)
Inflection strategies	Spanish *EE UU* ($<$ *Estados Unidos* 'United States')
Reduplication strategies	Indonesian *orang2* ($<$ *orangorang* 'people')
Other strategies	(none given)

Table 1: Abbreviation strategies queried in the survey, with characteristic examples.

3.1. Languages sampled

The languages, dialects, and scripts were selected to cover many language families and script types but also in support of internationalization and quality assurance efforts at Google. We decided to focus in some cases on multiple dialects of a particular "macrolanguage", or to target the multiple scripts used to write a certain language. In a slight abuse of terminology, we refer to the entries in our survey—a language or dialect, and the associated script—as *locales*. See Table 3 for a full list of locales. Some details of how locales were defined are discussed below.

For Arabic, which is both diglossic and pluricentric, the sample targeted both the Modern Standard literary standard as well as Egyptian, Gulf, and Magrebi dialects. For Gulf and Magrebi dialects, we target abbreviations in non-standard romanization. Hindi-Urdu was treated as two separate locales, as Hindi is written with a Brahmic alphasyllabary, and Urdu with a Perso-Arabic consonantal alphabet. Separate locales were used for the Brahmic (*Gurmukhi*) and Perso-Arabic (*Shahmukhi*) scripts used to write Punjabi in India and Pakistan, respectively, and for the non-standard romanizations of Bengali, Hindi, Marathi, and Urdu. Finally, European and Brazilian dialects of Portuguese were treated as separate locales.

3.2. Participants

At least three—though occasionally as many as nine—consultants gave judgments and examples for each locale. Consultants were recruited from a pool of professional annotators and were compensated for their time.

3.3. Instrument

The survey itself was conducted using Google Forms. Before the survey began, consultants were prompted to rate their proficiency with the target locale on a seven-point scale where 1 was labeled "limited proficiency" and 7 was labeled "native fluency". The median value for this proficiency score was 5, and no consultant scored their proficiency lower than 3. The survey consists of seven main questions, each asking whether the target locale uses a particular style of abbreviation. These strategies are listed in Table 1.

If the consultant answered yes, they were then asked to provide two or three relevant examples of that style (where each example includes the abbreviated form, the expanded form, and an English gloss). This initial taxonomy of style is based on the authors' own linguistic background and is not intended to exhaust the possibilities. Thus, in a final question, the consultant was asked whether they were aware of any styles of abbreviation not yet covered in the locale, and if so, were asked to provide examples of these styles. The text of the survey is reprinted in Appendix A.

3.4. Consensus

When consultants for a given locale disagreed as to whether a given abbreviation formation strategy was present in that locale, the first author manually enforced consensus across the responses for that locale. This was done by consulting the examples provided. If, for instance, only one of the three consultants provided an example of a given strategy, but the examples clearly illustrate the strategy, the omission by the other consultants was assumed to be accidental and the strategy is coded as present in the locale. However, if the examples provided were not of the relevant strategy, or were judged uninterpretable, they were discarded and the strategy was coded as absent.

4. Results

The survey received roughly 200 responses over 55 locales corresponding to 46 unique ISO 639-1 languages. Roughly half of these locales use non-Latin scripts. The associated language codes are adapted from ISO-639-1 with additional disambiguating information where necessary.

Disagreements between annotators were somewhat common, accounting for 39% of the re-

Abbreviation class	% attested
First-character abbreviations	90.6
Stump compounds	45.3
Truncations	56.6
Augmented truncations	30.2
Word-internal deletions	45.3
Inflection strategies	15.1
Reduplication strategies	3.8
Other strategies	17.0

Table 2: The percentage of locales attesting each of the eight abbreviation strategies.

sponses (not including "Other strategies") aggregated across locale. Table 2 provides the percentages of locales that attest (following the consensus procedure in subsection 3.4) each of our abbreviation classes. Table 3 in Appendix B provides the full post-consensus per-locale results. We give an impressionistic summary of the findings below.

As can be seen from Table 2, first-character abbreviations are present for most locales, but the other strategies in English are less commonly found. Truncations are the next most common strategy; these are attested in just over half of the locales. Inflection and reduplication marking are far less common than the other strategies. Roughly one out of six locales attest other strategies beyond the seven provided.

The locales for languages spoken in the Indian subcontinent—Assamese, Bengali, Gujarati, Hindi, Kannada, Malayalam, Marathi, Odia, Punjabi, Tamil, Telugu, and Urdu—make little use of abbreviation beyond first-character abbreviations (e.g., Kannada ಬಿಎಸ್ಎನ್ಎಲ್ ⟨bi-es-en-el⟩ 'Bharat Sanchar Nigam Limited'; note the failure to encode aspiration of the initial consonant), and some of these may be borrowed from English. This is marked insofar as these languages belong to two distinct language families (Indo-Aryan and Dravidian) and which may be written in Brahmic alphasyllabaries, Perso-Arabic consonantal alphabets, or Latin alphabetic transliterations.

Outside of the subcontinent, two other languages which make limited use of abbreviations are Arabic and Farsi. While these languages are unrelated, this may reflect a structural incompatibility between abbreviation formation and the Perso-Arabic script they share (also with Urdu, as mentioned above); this script is *defective* in the sense that short vowels are ordinarily omitted, and this in turn might limit the ability—or need—to form abbreviations via truncation or word-internal deletion. Something similar might be true of Korean; its *Hangul* writing system is roughly alphabetic, but symbols are organized into syllable-sized blocks called *jamo*, which might make it difficult to indicate abbreviations—particularly those which involve deletion of vowels—orthographically. Similarly, Japanese, which is written in a mixed—but predominantly syllabic—writing system, forms novel stump compounds from English phrases, as in セクハラ ⟨se-ku-ha-ra⟩ 'sexual harassment', but its writing system lacks any obvious way to represent the deletion of vowels. Amharic and Tigrinya, closely related Ethiopic languages written with the Geʿez alphasyllabary, also make limited used of abbreviation formation. Among Latin-script locales, Yorùbá and Zulu stand out for their limited use of abbreviations.

Relatively few locales surveyed make use of augmented truncations. Both dialects of Portuguese use *-ão* as an augment, as in *burgão* (< *hambúrguer* 'hamburger'). The Russian abbreviation презик 'condom' is formed from a truncation of презерватив suffixed with a -ик augment. Turkish uses truncation with an *-o* augment to form familiar forms of proper names. While it is not abbreviation per se, Indonesian uses truncation and infixation to derive informal forms of words. For example, *sepokat* is formed via truncation of word-final *u* in *sepatu* 'shoe' and infixation of *-ok-*.[7]

Inflection-marking strategies are overall rare. Belarusian, Polish, and Romanian generalize the character-doubling strategy found in Portuguese, Serbian and Spanish by reduplicating the entire abbreviation. In Polish, for example, the plural forms of *o.* (< *ojciec* 'father') and *prof.* (< *profesor* 'professor') are *oo.* and *prof. prof.*, respectively. In Slovenian, the title of someone with two doctoral degrees is abbreviated either as *ddr.* or *dr. dr.* Specific strategies for abbreviating reduplicated words are even less common. Indonesian and Vietnamese use the Arabic numeral *2* to indicate reduplication, and the 々 character serves the same purpose in Japanese.

Many consultants reported the presence of other strategies for abbreviation formation. Many locales report the use of a "mixed" strategies. For example, the derivation of Belarusian БелТА (< Беларускае Тэлеграфнае Агенцтва 'Belarusian Telegraph Agency'), Polish *PZMot* (< *Polski Związek Motorowy* 'Polish Automobile and Motorcycle Federation'), and Uzbek *SamDU* (< *Samarqand Davlat Universiteti* 'Samarkand State University') seems to combine truncation and first-character abbreviation. Bulgarian forms abbreviations by deleting a contiguous sequence of word-internal segments, marking the deleted span with a hyphen—e.g., у-ще (< училище 'school')—and a similar strategy is found in Hebrew.

[7]This process is apparently borrowed from a thieves' argot. A similar process, the infixation of *-iz-* in African-American Vernacular English, is described by Gil Scott-Heron in his spoken-word piece "The Ghetto Code".

5. Conclusions

We have presented (and publicly released) results from a survey over diverse languages and scripts regarding abbreviatory devices in the writing systems. Explicit examples in a language of specific attested abbreviation strategies can help in developing covering grammars or similar pattern-matching methods (e.g., Gorman and Sproat, 2016; Sproat and Jaitly, 2017; Zhang et al., 2019) to find possible abbreviations and candidate expansions in raw text, en route to building abbreviation expansion engines. In future work, we intend to mine web text to identify additional examples of attested patterns.

Acknowledgments

Thanks to Alexander Gutkin, Christo Kirov, and Richard Sproat; and to the many annotators upon whom this project depended.

References

Timothy Baldwin, Young-Bum Kim, Marie Catherine de Marneffe, Alan Ritter, Bo Han, and Wei Xu. 2015. Shared tasks of the 2015 Workshop on Noisy User-generated Text: Twitter lexical normalization and named entity recognition. In *Proceedings of the Workshop on Noisy User-generated Text*, pages 126–136.

Garland Cannon. 1989. Abbreviations and acronyms in English word-formation. *American Speech*, 64(2):99–127.

Grzegorz Chrupała. 2014. Normalizing tweets with edit scripts and recurrent neural embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 680–686.

David Crystal. 2001. *Language and the Internet*. Cambridge University Press.

David Crystal. 2008. *Txtng: The Gr8 Db8*. Oxford University Press.

Peter Ebden and Richard Sproat. 2015. The Kestrel TTS text normalization system. *Natural Language Engineering*, 21(3):1–21.

Daan van Esch and Richard Sproat. 2017. An expanded taxonomy of semiotic classes for text normalization. In *Proceedings of INTERSPEECH*, pages 4016–4020.

Kyle Gorman, Christo Kirov, Brian Roark, and Richard Sproat. 2021. Structured abbreviation expansion in context. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 995–1005.

Kyle Gorman and Richard Sproat. 2016. Minimally supervised number normalization. *Transactions of the Association for Computational Linguistics*, 4:507–519.

Hans Marchand. 1969. *The Categories and Types of Present-Day English Word-Formation*, 2nd edition. Beck.

Deana Pennell and Yang Liu. 2010. Normalization of text messages for text-to-speech. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4842–4845.

Ernest Pusateri, Bharat Ram Ambati, Elizabeth Brooks, Ondrej Platek, Donald McAllaster, and Venki Nagesha. 2017. A mostly data-driven approach to inverse text normalization. In *Proceedings of INTERSPEECH*, pages 2784–2788.

Brian Roark and Richard Sproat. 2014. Hippocratic abbreviation expansion. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 364–369.

Richard Sproat, Alan W. Black, Stanley Chen, Shankar Kumar, Mari Ostendorf, and Christopher Richards. 2001. Normalization of non-standard words. *Computer Speech & Language*, 15(3):287–333.

Richard Sproat and Navdeep Jaitly. 2017. An RNN model of text normalization. In *Proceedings of INTERSPEECH*, pages 754–758.

Hao Zhang, Richard Sproat, Axel H. Ng, Felix Stahlberg, Xiaochang Peng, Kyle Gorman, and Brian Roark. 2019. Neural models of text normalization for speech applications. *Computational Linguistics*, 45(2):293–337.

Piotr Żelasko. 2018. Expanding abbreviations in a strongly-inflected language: are morphosyntactic tags sufficient? In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation*, pages 1880–1884.

A. Survey questions

1. Does the target language use abbreviations formed from the first character of each word in a phrase (e.g., “NATO” $<$ “North Atlantic

Treaty Organization", "CIA" < "Central Intelligence Agency")? [**If yes:** Please provide 2–3 examples of abbreviations formed from the first character of each word in a phrase in the target language, giving the abbreviated form, the full/expanded form, and an English gloss. (You are also welcome to link to any relevant discussions of this phenomenon.)]

2. Does the target language use abbreviations formed from the first syllable of each word in a phrase (e.g., "FiDi" < "Financial District", "ForEx" < "foreign exchange")? [**If yes:** Please provide 2–3 examples of abbreviations formed from the first syllable of each word in a phrase in the target language, giving the abbreviated form, the full/expanded form, and an English gloss. (You are also welcome to link to any relevant discussions of this phenomenon.)]

3. Does the target language use abbreviations formed by truncating characters at the ends of words (e.g., "Col." < "Colonel", "Ave." < "Avenue")? [**If yes:** Please provide 2–3 examples of abbreviations formed by truncating characters at the ends of words in the target language, giving the abbreviated form, the full/expanded form, and an English gloss. (You are also welcome to link to any relevant discussions of this phenomenon.)]

4. Does the target language use abbreviations formed by truncating characters at the ends of words and then adding "augment" suffixes (e.g., Australian English "footie" < "football", with an -ie augment, "ambo" < "ambulance" with an -o augment)? [**If yes:** Please provide 2–3 examples of abbreviations formed by truncating characters at the ends of words and then adding "augment" suffixes in the target language, giving the abbreviated form, the full/expanded form, and an English gloss. (You are also welcome to link to any relevant discussions of this phenomenon.)]

5. Does the target language use abbreviations formed by deleting characters (e.g., particularly vowels) from the middle of words (e.g., "Blvd." < "Boulevard", "Sgt." < "Sergeant")? [**If yes:** Please provide 2–3 examples of abbreviations formed by deleting characters from the middle of words in the target language, giving the abbreviated form, the full/expanded form, and an English gloss. (You are also welcome to link to any relevant discussions of this phenomenon.)]

6. Does the target language use any orthographic tricks to mark the inflection of abbreviations (e.g., Spanish "EE UU." < "Estados Unidos" 'United States', with the doubling used to indicate that the abbreviation is plural)? [**If yes:** Please provide 2–3 examples of inflected abbreviations in the target language, giving the abbreviated form, the full/expanded form, and an English gloss. (You are also welcome to link to any relevant discussions of this phenomenon.)]

7. Does the target language use any particular orthographic tricks to abbreviate reduplication (e.g., Indonesian "orang2" < "orango-rang" 'people', with "2" used to indicate reduplication)? Answer "No" if the target language does not have productive reduplication. [**If yes:** Please provide 2–3 examples of abbreviations for reduplication in the target language, giving the abbreviated form, the full/expanded form, and an English gloss. (You are also welcome to link to any relevant discussions of this phenomenon.)]

8. Does the target language use any other style of abbreviation not yet covered? [**If yes:** Please provide 2–3 examples of these other style or styles, giving the abbreviated form, the full/expanded form, and an English gloss. (You are also welcome to link to discussions of other styles of abbreviation in the target language.)]

B. Consensus locale results

Table 3 presents the consensus results for each of the locales surveyed.

Code	Locale	*NATO*	*FiDi*	*Col.*	*footie*	*Blvd.*	*EE UU*	*orang2*	Other
am	Amharic	✓	✗	✗	✗	✗	✗	✗	✗
ar-eg	Arabic (Egyptian)	✓	✗	✗	✗	✗	✗	✗	✗
ar-gu	Arabic (Gulf, Latin)	✓	✗	✗	✗	✗	✗	✗	✗
ar-ml	Arabic (Magrebi, Latin)	✓	✗	✗	✗	✗	✗	✗	✗
ar-ms	Arabic (Modern Standard)	✓	✗	✗	✗	✗	✗	✗	✗
as	Assamese	✓	✗	✗	✗	✗	✗	✗	✗
be	Belarusian	✓	✓	✓	✗	✓	✓	✗	✓
bg	Bulgarian	✓	✓	✓	✗	✗	✗	✗	✗
bn	Bengali	✓	✓	✓	✗	✓	✗	✗	✗
bn-la	Bengali (Latin)	✓	✓	✓	✓	✓	✗	✗	✗
en	English	✓	✓	✓	✓	✓	✗	✗	✗
es	Spanish	✓	✗	✗	✗	✓	✓	✗	✓
et	Estonian	✓	✓	✓	✗	✓	✗	✗	✗
fa	Farsi	✓	✗	✗	✗	✗	✗	✗	✗
fr	French	✓	✓	✓	✗	✗	✗	✗	✓
gu	Gujarati	✓	✗	✗	✓	✗	✗	✗	✗
ha	Hausa	✓	✗	✓	✓	✓	✗	✗	✓
he	Hebrew	✓	✓	✓	✓	✗	✗	✗	✓
hi	Hindi	✓	✗	✗	✓	✗	✗	✗	✗
hi-la	Hindi (Latin)	✓	✗	✓	✓	✗	✗	✗	✗
hy	Armenian	✓	✓	✓	✓	✓	✗	✗	✓
id	Indonesian	✓	✓	✓	✓	✓	✗	✓	✓
it	Italian	✓	✗	✓	✓	✓	✗	✗	✗
ja	Japanese	✓	✗	✗	✗	✗	✗	✓	✗
ka	Georgian	✓	✓	✓	✓	✓	✗	✗	✗
kn	Kannada	✓	✗	✗	✗	✓	✗	✗	✗
ko	Korean	✗	✓	✗	✗	✗	✗	✗	✗
lt	Lithuanian	✓	✓	✗	✗	✓	✗	✗	✗
lv	Latvian	✓	✗	✓	✗	✓	✗	✗	✗
ml	Malayalam	✓	✗	✗	✗	✗	✗	✗	✗
mr	Marathi	✓	✗	✗	✗	✗	✗	✗	✗
mr-la	Marathi (Latin)	✗	✗	✗	✗	✗	✗	✗	✗
nl	Dutch	✓	✓	✓	✗	✓	✗	✗	✗
or	Odia	✓	✓	✗	✗	✗	✗	✗	✗
pa-gu	Punjabi (Gurmukhi)	✗	✗	✗	✗	✗	✗	✗	✗
pa-sh	Punjabi (Shahmukhi)	✓	✗	✗	✗	✗	✗	✗	✗
pl	Polish	✓	✓	✓	✗	✓	✓	✗	✓
pt-br	Portuguese (Brazilian)	✓	✓	✓	✓	✓	✗	✗	✗
pt-pt	Portuguese (European)	✓	✓	✓	✓	✓	✓	✗	✗
ro	Romanian	✓	✓	✓	✗	✗	✓	✗	✗
ru	Russian	✓	✓	✓	✓	✓	✗	✗	✓
sl	Slovenian	✓	✓	✓	✓	✓	✗	✗	✗
sq	Albanian	✓	✗	✓	✗	✓	✓	✗	✗
sr	Serbian	✓	✓	✓	✗	✗	✓	✗	✗
sw	Swahili	✓	✗	✓	✗	✓	✗	✗	✗
ta	Tamil	✓	✗	✓	✗	✗	✗	✗	✗
te	Telugu	✓	✗	✓	✗	✗	✗	✗	✗
ti	Tigrinya	✗	✗	✓	✗	✗	✗	✗	✗
tr	Turkish	✓	✓	✓	✓	✓	✗	✗	✗
ur	Urdu	✗	✗	✗	✗	✗	✗	✗	✗
ur-la	Urdu (Latin)	✓	✗	✗	✗	✗	✗	✗	✗
uz	Uzbek	✓	✓	✓	✗	✓	✗	✗	✗
vi	Vietnamese	✓	✓	✗	✓	✓	✓	✗	✗
yo	Yorùbá	✓	✗	✗	✗	✗	✗	✗	✗
zu	Zulu	✗	✗	✓	✗	✓	✗	✗	✗

Table 3: Summary of the abbreviation strategies attested in each locale after consensus.

Now You See Me, Now You Don't: 'Poverty of the Stimulus' Problems and Arbitrary Correspondences in End-to-End Speech Models

Daan van Esch
Google Speech
1600 Amphitheatre Parkway, Mountain View, CA 94043
dvanesch@google.com

Abstract

End-to-end models for speech recognition and speech synthesis have many benefits, but we argue they also face a unique set of challenges not encountered in conventional multi-stage hybrid systems, which relied on the explicit injection of linguistic knowledge through resources such as phonemic dictionaries and verbalization grammars. These challenges include handling words with unusual grapheme-to-phoneme correspondences, converting between written forms like '12' and spoken forms such as 'twelve', and contextual disambiguation of homophones or homographs. We describe the mitigation strategies that have been used for these problems in end-to-end systems, either implicitly or explicitly, and call out that the most commonly used mitigation techniques are likely incompatible with newly emerging approaches that use minimal amounts of supervised audio training data. We review best-of-both-world approaches that allow the use of end-to-end models combined with traditional linguistic resources, which we show are increasingly straightforward to create at scale, and close with an optimistic outlook for bringing speech technologies to many more languages by combining these strands of research.

Keywords: speech recognition, speech sythesis, end-to-end modeling, text normalization, pronunciation modeling

1. Introduction

In recent years, so-called 'end-to-end' models have become increasingly popular for both automatic speech recognition (ASR) and text-to-speech (TTS) applications. The precise meaning of 'end-to-end' is not exactly defined, and there are many variations on the theme, but in the case of ASR, end-to-end models are typically understood to be all-neural systems that take in audio features and emit some subword-level unit like bytes (Li et al., 2019), graphemes, or wordpieces directly (Prabhavalkar et al., 2017). Such all-neural systems side-step the need for a manually-curated phonemic dictionary of the target language (Sainath et al., 2018; Kim et al., 2020), and contrast with 'conventional' or 'hybrid' approaches, which consist at least partially of non-neural components, like a phonemic lexicon finite-state transducer (Mohri et al., 2002) or hand-written verbalization grammars (Sak et al., 2013) to turn words like 'twelve' into '12'. Similarly, in TTS, end-to-end architectures like Tacotron (Wang et al., 2017) can take in graphemes and emit audio, again without going through intermediate phases that are common in conventional TTS systems, such as text normalization to turn input like '12' into 'twelve' (Sproat et al., 2001), and grapheme-to-phoneme conversion, typically employing a combination of phonemic dictionaries and machine-learning models (Bisani and Ney, 2008).

In conventional speech processing systems, with multiple individual components involved, jointly optimizing over the entire system from start to finish is hard, and errors may compound, which can be detrimental to quality (Wang et al., 2017). By design, all-neural end-to-end approaches do not have this issue, while also being much simpler to train (Chiu et al., 2017), as well as being significantly smaller in terms of disk size, enabling deployment of high-quality models on devices like smartphones (Kim et al., 2020). In addition, being able to avoid the need for injecting linguistic knowledge, such as phonemic dictionaries or verbalization grammars, is frequently considered to be an advantage of end-to-end modeling approaches—for example, (Wang et al., 2017) points out that such components require 'extensive domain expertise and are laborious to design', while an end-to-end approach 'alleviates the need for laborious feature engineering'. Similarly, (Sotelo et al., 2017) argues that end-to-end modeling for TTS 'eliminates the need for expert linguistic knowledge, which removes a major bottleneck in creating synthesizers for new languages'.

Unarguably, developing such linguistic resources can take a non-negligible amount of effort, even despite much progress on tools and methodologies that can help alleviate this burden (Kim and Snyder, 2013; Rutherford et al., 2014; Deri and Knight, 2016; Lee et al., 2020; Ritchie et al., 2019; Bleyan et al., 2019; Ritchie et al., 2020). And it is, of course, desirable to mitigate bottlenecks in developing speech systems in whatever way possible: if it were indeed possible to completely avoid the need for linguistic resources while still building systems that are in every way just as capable of handling ASR or TTS tasks as conventional systems, or even better at doing so (as suggested by Sainath et al. (2018)), that would be excellent. However, this may be infeasible, as the relationship between the spoken and the written form of human languages is typically far from straightforward: such correspondences frequently turn out to be full of entirely arbitrary phenomena that cannot be derived, or that would be very difficult to derive, through generalization from a randomly selected large set of training data in the target language.

We provide an overview of such challenging correspondences, and argue that this area has not been receiving enough attention and analysis in the literature on end-to-end systems, calling for more research and analysis along the lines of (Fong et al., 2019; Taylor and Richmond, 2019) to investigate how well end-to-end speech modeling approaches are equipped to deal with such problems. We focus on a practical description of the problem space and common mitigation techniques more so than empirical experiments, since the ability of end-to-end speech systems to handle such arbitrary correspondences will differ depending on factors such the composition of the training data, plus model architecture and size. We argue that these issues will become all the more relevant as end-to-end modeling approaches are adopted that use only minimal amounts of supervised target-language training data: for example, (Baevski et al., 2020; He et al., 2021) report impressive progress on extending speech technologies to new languages using just minutes of transcribed target-language audio, but the resulting models are likely to struggle with such arbitrary correspondences when no linguistic resources are used.

2. Poverty of the Stimulus

As an example, assume for the sake of argument that an end-to-end model's training data contains *all* numbers in the English language except for '12' (and except numbers of which '12' is a constituent, like '112' as 'one hundred and twelve'). Given this data, this hypothetical model would be unable to know that 'twelve' should be transcribed as '12' in ASR tasks, or that '12' can be read as 'twelve' in TTS tasks. This is because there is no possible generalization that would let the model determine that the written form '12' corresponds to the spoken form 'twelve'. To avoid generalizing to something incorrect like 'twoteen' (by analogy with 'fourteen', 'sixteen', 'seventeen', and so on) or 'ten-two' (by analogy with 'twenty-two', 'thirty-two', and so on), an end-to-end model *needs* to observe at least one example of this arbitrary correspondence in its training data.

This problem is by no means limited to '12' alone, of course: even just among English numbers between 10 and 20, cases like '11' and 'eleven', '13' and 'thirteen', and '15' and 'fifteen' are not quite straightforwardly generalizable either. More generally, such examples are somewhat reminiscent of what's known in linguistics as the 'poverty of the stimulus', which is the argument that children are not exposed to enough information to correctly induce the rules of their native language, and that it must therefore be the case that the human brain must contain some form of innate knowledge about how languages work (Laurence and Margolis, 2001). This debate is far from settled in linguistics, but what is relevant to us here is the idea of analyzing a learner's input to understand the limits of generalization based on such training data, and to test how well learners generalize at various points of the learning process.

We believe end-to-end speech processing would benefit from (1) taking into account the limits of generalization when it comes to correspondences between spoken and written forms of language, (2) understanding to what extent the training data used for a given model can theoretically allow it to learn these correspondences, (3) evaluating the degree to which this process is successful in practice, e.g. through separate test sets for various types of numeric sequences, or words with unusual grapheme-to-phoneme correspondences, and (4) taking steps to make available any missing linguistic knowledge to the model in the next round of training, or at inference time.

2.1. Semiotic Classes: Numbers, Times, etc.

In our hypothetical example above, '12' would need to be observed in the training data for the model to learn this unusual and arbitrary correspondence. Including it in the training data, then, would be a natural solution. And indeed, our argument is not that an end-to-end model could never learn such edge cases at all—we simply observe, in a flavor of the poverty-of-the-stimulus argument, that the system *must* have observed such idiosyncratic cases at least once to be able to learn them.

Of course, this does raise the question of what needs to be included in the training data. Some correspondences will be generalizable: for example, a system that observed in its training data all the numbers between '20' and '40' apart from '33' should be able to generalize correctly and produce 'thirty-three' for '33', following the pattern of first verbalizing the decade form ('twenty' or 'thirty') and then using the regular cardinal number 'three' (as in 'thirty-one', 'thirty-two', and so on). In other words, some examples are entirely arbitrary and idiosyncratic, and need to be specifically covered individually, while for others, generalization is an option as long as sufficient information is available from which to generalize.

Now, even if a model observes such an entirely idiosyncratic case such as '12' only once at training time, the model may not remember this correspondence; to our knowledge there has been no research determining if there is a threshold of occurrences that is sufficient to teach an end-to-end system a given correspondence. Presumably the degree of arbitrariness and frequency both play a role, as does the general model architecture, but this seems like a rich area for analysis. However, an end-to-end model would at least theoretically stand a chance to get '12' right if it was included even just once in its training data; if it never observed this case at all, it would stand no chance.

If the issues were limited to cases like like '10', '11' and '12', this would perhaps pose only a limited problem that could be easily addressed by including relevant data at training time. However, even for simple cardinal numbers in counting forms like 'one', 'two', 'three', a relatively complex induction needs to be done to derive the correct correspondences even just for forms between 1 and 999, as shown by Ritchie et al. (2019); Gorman and Sproat (2016). These inductions differ in complexity from one language to another, but are rarely straightforward. In the extreme, for some languages, the only option for getting the numbers between 1 and 100 right appears to be explicitly enumerating every single form (Gorman and Sproat, 2016). This would imply that an end-to-end speech system would also need to observe every single form in its training data at least once.

Cardinal numbers are, unfortunately, perhaps among the *easier* correspondences to learn. There are many classes of tokens for which there are non-trivial correspondences between written and spoken forms of human language, and they appear in mostly any language; see e.g. van Esch and Sproat (2017) which provides an overview of these 'semiotic classes', ranging from phone numbers like '1-800-GOOG411' to times like '10:15' ('ten fifteen' or 'a quarter past ten'), and from measures like '10km' to currency amounts like 'HK$300'.

For end-to-end speech models, learning such correspondences is difficult, with low accuracy rates unless special measures are taken (Peyser et al., 2019). In fact, as Sproat and Jaitly (2017); Zhang et al. (2019) show, handling semiotic classes is difficult even for standalone text-to-text neural networks that are dedicated entirely to transforming between spoken-domain text strings like 'twelve' and written-domain text strings like '12'. Sproat and Jaitly (2017); Zhang et al. (2019) point out that even such text-to-text models frequently make so-called 'silly errors' like verbalizing '16GB' into 'sixteen hertz' instead of 'sixteen gigabytes'—even though for these networks, large amounts of relevant data were available at training time, based on which the correct behavior *could* have been inferred.

2.2. Normal Words: Names, Loanwords, Acronyms, etc.

Beyond semiotic-class tokens like '12', '1-800-GOOG411', '10:15', 'HK$300', and '16GB', there are also countless examples of English words with idiosyncratic grapheme-to-phoneme mappings. For example, the pronunciation of the English word 'Worchestershire' is relatively idiosyncratic, consisting of only three syllables. Put simply, the rules of English orthography do not map one-to-one onto English pronunciations—and this is the case in many of the world's languages (though not everywhere). The complexity of various orthographic systems can be measured (van den Bosch et al., 1994), and different orthographic systems are known to have different degrees of orthographic transparency (Katz and Frost, 1992). In practice, this means that in some languages, the correspondences between spoken and written forms will be harder to learn than in others.

Indeed, cross-language comparisons of grapheme-to-phoneme (G2P) conversion models such as (van Esch et al., 2016; Lee et al., 2020) show widely different accuracy rates across languages. While the accuracy metrics achieved by G2P models also depend on the amount of training data available for the target language, as well as factors such as the model architecture, there is unmistakably an impact from the degree of orthographic transparency in each language. Some languages, like Spanish, have a reasonably transparent orthography, and G2P accuracy rates are usually high; languages like English, on the other hand, feature large numbers of idiosyncratic cases, which are much more challenging or even impossible for a G2P model to predict based on the training data, leading to lower accuracy rates.

Such challenging grapheme-to-phoneme correspondences are known to impact the quality of end-to-end speech models: for example, Taylor and Richmond (2019); Fong et al. (2019) show that end-to-end TTS models struggle to generate correct audio for words with irregular or idiosyncratic G2P correspondences. End-to-end ASR systems face similar struggles (Kim et al., 2020; Prabhavalkar et al., 2017), although to our knowledge the issue has not been analyzed in detail. Unusual grapheme-to-phoneme correspondences appear in many types of words, including in place names like ‘Worchestershire’, names of people and businesses (Rutherford et al., 2014), names of artists (like ‘P!nk’, where the ‘!’ stands for an ‘i’, or ‘deadmau5’, read as ‘dead mouse’), and loanwords (which may retain the original spelling from their source language, as in ‘restaurant’ or ‘La Jolla’). Sometimes, otherwise entirely unremarkable nouns suddenly involve an unpredictable correspondence, as in ‘sword’, the only word in the English language where the grapheme ‘w’ is silent in the onset cluster ‘sw’: compare, for example, ‘swam’, ‘sweep’, and ‘swore’. Highly frequent words may also have unusual grapheme-to-phoneme correspondences, like English ‘one’. And letter sequences (Sproat and Hall, 2014) such as ‘NASA’ (read as a word) and ‘C-SPAN’ (partially read as a word, partially as a letter) present their own idiosyncrasies—not to mention borrowed letter sequences, such as ‘BBC’, which is read letter-by-letter using English letter pronunciations in many European languages.

As with semiotic-class tokens, it can range from challenging to impossible for an end-to-end speech model to predict the correct grapheme-to-phoneme correspondence for a given word, depending on the degree of arbitrariness of the relationship, the training data, and the model’s generalization ability. According to the Census Bureau, there are tens of thousands of geographical names in the United States alone. Many of them are likely reasonably amenable to generalization, but others (like ‘La Jolla’) will not be, and must be observed in the training data for an end-to-end model to learn them. In the extreme, cases like ‘deadmau5’ are presumably sufficiently idiosyncratic as to be impossible for an end-to-end system to predict correctly through generalization from any other training data.

2.3. Homophones, Homographs, and Context

For both semiotic-class tokens and normal words, another issue can cause further challenges for ASR, namely homophony—words or phrases that sound the same, but have different spellings depending on their meaning and context. For example, ‘three eleven’ could be written as ‘3:11’ (as a time) or ‘3/11’ (as a date), and ‘Xanh’ (a popular restaurant in Mountain View, California, which unfortunately closed after the pandemic) shares its pronunciation with ‘sun’. As an extreme example, if these terms were only ever observed in isolation at training time, the system would find it challenging to determine that it should emit ‘dinner at Xanh in Mountain View’ (not ‘sun’) but ‘the sun is shining’ (not ‘Xanh’).

In TTS applications, homographs, or words that are spelled the same but have different pronunciations depending on context, pose similar problems: ‘Houston, Texas’ is pronounced differently than ‘Houston Street, New York City’ (which is pronounced like ‘how-ston’ not ‘hew-ston’), but again, if the term ‘Houston’ was only ever observed in isolation, the model would struggle to decide which of the two pronunciations to use based on inference-time context—assuming, of course, that both pronunciations were even included in the training data.

3. Mitigation Techniques

The existence of such arbitrariness is not an argument against end-to-end modeling: our goal has only been to point out that it is impossible for an end-to-end model to correctly predict an entirely arbitrary phenomenon that it has not observed at training time—and that even for slightly less arbitrary phenomena, such models may struggle. But given the benefits of end-to-end modeling, it is clearly desirable to see if these challenges can be mitigated within the end-to-end paradigm.

Before discussing mitigation strategies, one question that may come to mind is whether any mitigation is in fact needed at all: one might argue that handling these correspondences can simply be called out-of-scope entirely. For example, an ASR system could simply emit ‘twelve’ instead of ‘12’, and a TTS system could simply require that only forms like ‘twelve’ are used in any input text. However, in a real-world system this is typically infeasible or impractical, given that TTS applications are generally expected to be able to handle generic written-domain text, and given that downstream processing of ASR transcriptions generally also relies on written forms like ‘11:15’—for example, in conversational voice assistants that need to identify times in transcribed spoken commands. Taking this position is even harder in the case of words with unusual grapheme-to-phoneme relationships: it would be hard to argue that general-purpose ASR or TTS systems do not need to correctly pronounce or transcribe phrases like ‘La Jolla’.

First, we recommend setting up evaluation sets that specifically aim to measure ASR or TTS quality for different categories of arbitrary correspondences, such as words with unusual grapheme-to-phoneme relationships, and different types of semiotic classes, as in Peyser et al. (2019). Such sets will help us understand the extent to which these problems appear for a given model. The question then becomes how to maximize accuracy for these cases.

3.1. Large Training Data Sets

Ensuring that end-to-end systems see sufficient data to correctly generalize all generalizable correspondences, and to learn even the most arbitrary cases, is one possible approach. This does pose some practical problems, since there will be *many* words that are affected (especially in languages like English, with its opaque orthography). But as one increases the size of the training data, more correspondences will be covered, mitigating the problem to some extent—and with the abundance of data in high-resource languages, the problem may even be invisible entirely unless specific evaluations are done, as in Peyser et al. (2019).

Theoretically, one could simply collect recordings of *all* normal words and semiotic-class tokens in the target language, but this would clearly be very time-consuming, and it is not clear that it poses less of a bottleneck than creating verbalization grammars and phonemic dictionaries. In the extreme, it is arguably impossible due to the infinite amount of e.g. cardinal numbers, but at at any rate, recording many hundreds of thousands words and phrases would be challenging even for ASR, where training data can be gathered from many speakers through platforms such as Hughes et al. (2010); for TTS, where high-quality single-speaker recordings have typically been required, it would be entirely impractical.

In addition to practical factors that make adding more training data a less-than-desirable mitigation strategy, recent work also suggests that reasonable levels of ASR or TTS quality can be obtained by using only an hour or less of supervised target-language audio (Baevski et al., 2020; He et al., 2021), combined with self-supervised learning techniques and/or multilingual modeling. Such work is incredibly promising for addressing the single biggest bottleneck in bringing speech technologies to more languages, namely the scarcity of supervised training data, but it seems vanishingly unlikely that 40 seconds of target-language audio (as in He et al. (2021)) could contain sufficient information to learn all relevant arbitrary correspondences for the target language.

In some cases, multilingual modeling may help, e.g. in predicting that '24' should be verbalized as 'vingt-quatre' in French, following the English pattern. But equally, multilingual modeling may be ineffective or even harmful for this problem, e.g. when mixing English with German, where the correct verbalization of '24' is not 'zwanzig-vier' (literally 'twenty-four'), but 'vier-und-zwanzig' ('four-and-twenty').

3.2. Supplementing Training Data with Synthetic Audio

For ASR, another technique is to use TTS to generate synthetic data to supplement the training data (Rosenberg et al., 2019): for example, Peyser et al. (2019) showed that if a target-language TTS system is available, it can be used to generate transcribed-audio training examples for cases like '12' and 'twelve' at very large scale. However, such approaches still requires the creation of some kind of verbalization grammar (Sak et al., 2013; Ritchie et al., 2019, 2020) to provide the correspondences between the written-domain forms (like '12') which would serve as the ASR training target, and the spoken-domain forms (like 'twelve') which would be passed into the TTS system. And unless the target-language orthography is extremely transparent, the TTS system itself will likely require a phonemic dictionary (recall cases like 'one') in order for the synthetic audio it generates to have the correct pronunciation. Similar synthetic-audio approaches can be employed for phrases like 'La Jolla' with challenging grapheme-to-phoneme correspondences, but again this would require a phonemic dictionary to drive the generation of accurate synthetic audio. In other words, such synthetic-data approaches still require an investment in linguistic resources that is no different from the investments needed to build the linguistic components of conventional, non-end-to-end systems.

3.3. Secondary Models

Yet another class of mitigation techniques involves combining secondary models with the original end-to-end model. For example, fusion techniques are commonly used to connect end-to-end ASR models with neural text-only language models to cover phrases that were not observed in the original training data (Kim et al., 2020). While it seems reasonable that external language models can help with contextual disambiguation ('Xanh' vs 'sun'), their effect for words with unusual grapheme-to-phoneme or arbitrary verbalization correspondences is unclear and requires further research; they are unlikely to be a panacea, especially for highly idiosyncratic cases. In another example, Serrino et al. (2019) describes a module that allows for the use of phonemic dictionaries to correct misrecognitions from the upstream ASR system, but again at the cost of requiring linguistic resources. In both ASR and TTS, secondary neural models can also be used for normalizing semiotic-class tokens before or after the core end-to-end model, as in Zhang et al. (2019); Peyser et al. (2019); such models do, however, typically require large amounts of text-to-text training data, as well as covering grammars, both of which again require linguistic expertise.

3.4. Combining End-to-End and Conventional Approaches

It is also possible to combine the best of both worlds, so to speak, by training models using the end-to-end paradigm which do however still use phonemes as an input or output unit: one recent example of this is the Hybrid Autoregressive Transducer (HAT) (Variani et al., 2020) for ASR, which combines end-to-end models that output phoneme units with a traditional finite-state transduction decoding graph that uses a phonemic dictionary and verbalization grammars. Similarly, in TTS, end-to-end models can simply take phonemes produced by a conventional text normalization front-end as input, e.g. as in Skerry-Ryan et al. (2018); Yasuda et al. (2020). In a related approach, (Kastner et al., 2019) describes an end-to-end TTS model that allows the mixing of graphemes and phonemes in inference-time inputs, allowing per-example control through phonemic specifications where needed—but then the question becomes how to decide when such control should be exercised. Such best-of-both-worlds approaches share one commonality: they still require the same linguistic components as non-end-to-end systems.

4. Conclusions

End-to-end speech models face challenges when it comes to handling words with unusual grapheme-to-phoneme correspondences (e.g. place names and loanwords) and semiotic classes (e.g. numbers and time expressions), because there is a large amount of arbitrariness in the correspondences between spoken and written forms of human language, and because training data suffers from poverty-of-the-stimulus issues. Traditional speech systems solved these challenges through the explicit injection of linguistic knowledge, e.g. through phonemic dictionaries or verbalization grammars. With thousands of languages spoken in our world, and very few of them covered by speech technologies today, it would be great if we did not need to curate such resources for every language, but this is unlikely to be possible, given that the mitigation strategies we reviewed still require similar amounts of linguistic resources, as we have discussed.

Importantly, we called out that the mitigation strategy employed (mostly implicitly) for many end-to-end systems will no longer work as end-to-end approaches take hold that use relatively small amounts of supervised training data. These approaches rely heavily on self-supervised learning and multilingual modeling—an exciting development that promises to help bring speech technologies to many more languages. At the same time, as we have seen, these methods will likely need to be combined with synthetic-data approaches (in ASR), or best-of-both-world architectures, like the use of phonemes produced by a conventional text normalization front-end as the input unit to end-to-end TTS models, or like HAT (Variani et al., 2020) in ASR.

Fortunately, much work has been done to make creating linguistic resources like phonemic dictionaries and verbalization grammars for new languages easier than ever (Kim and Snyder, 2013; Rutherford et al., 2014; Deri and Knight, 2016; Lee et al., 2020; Ritchie et al., 2019; Bleyan et al., 2019; Ritchie et al., 2020), leading us to be optimistic about the opportunities for bringing high-quality ASR and TTS systems to more languages by combining conventional linguistic resources with innovative modeling approaches that require little supervised audio training data.

5. Acknowledgements

Many thanks to Richard Sproat, Trevor Strohman, Tara Sainath, Jonas Fromseier Mortensen, Pedro Moreno, and Jeremy O'Brien for many lively water cooler conversations on this topic over the years, and for their thoughtful feedback.

6. Bibliographical References

Solomon Teferra Abate, Martha Yifiru Tachbelie, and Tanja Schultz. 2020. Multilingual acoustic and language modeling for ethio-semitic languages. In *Proceedings of Interspeech 2020*, pages 1047–1051.

Oliver Adams et al. 2019. Massively multilingual adversarial speech recognition. In *Proceedings of NAACL 2019*, pages 96–108, Minneapolis, Minnesota. Association for Computational Linguistics.

Oliver Adams et al. 2021. User-friendly automatic transcription of low-resource languages: Plugging ESPnet into Elpis. In *Proceedings of ComputEL-4*.

Adam Albright. 2009. Lexical and morphological conditioning of paradigm gaps. *Modeling ungrammaticality in optimality theory*, pages 117–164.

Rosana Ardila et al. 2020. Common Voice: A massively-multilingual speech corpus. In *Proceedings of LREC 2020*, pages 4218–4222, Marseille, France. ELRA.

Alexei Baevski et al. 2020. wav2vec 2.0: A framework for self-supervised learning of speech representations. *arXiv:2006.11477*.

Laurent Besacier et al. 2014. Automatic speech recognition for under-resourced languages: A survey. *Speech Communications*, 56:85–100.

Maximilian Bisani and Hermann Ney. 2008. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Commun.*, 50(5):434–451.

David Blachon et al. 2016. Parallel speech collection for under-resourced language studies using the Lig-Aikuma mobile device app. In *Proceedings of SLTU 2016*, Yogyakarta, Indonesia.

Harry Bleyan et al. 2019. Developing pronunciation models in new languages faster by exploiting common grapheme-to-phoneme correspondences across languages. In *Proceedings of Interspeech 2019*.

Nicholas Buckeridge and Ben Foley. 2020. Scaling language data import/export with a data transformer interface. In *Proceedings of SLTU-CCURL 2020*, pages 121–125, Marseille, France. ELRA.

Isaac Caswell et al. 2020. Language ID in the wild: Unexpected challenges on the path to a thousand-language web text corpus. In *Proceedings of COLING 2020*.

Isaac Caswell et al. 2021. Quality at a glance: An audit of web-crawled multilingual datasets. *arXiv:2103.12028*.

Tania Chakraborty et al. 2021. A large scale low-resource pronunciation data set mined from Wikipedia. *arXiv:2101.11575*.

Po-Han Chi et al. 2021. Audio ALBERT: A lite BERT for self-supervised learning of audio representation. *arXiv:2005.08575*.

Chung-Cheng Chiu, Tara N. Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J. Weiss, Kanishka Rao, Katya Gonina, Navdeep Jaitly, Bo Li, Jan Chorowski, and Michiel Bacchiani. 2017. State-of-the-art speech recognition with sequence-to-sequence models. *CoRR*, abs/1712.01769.

Mason Chua et al. 2018. Text normalization infrastructure that scales to hundreds of language varieties. In *Proceedings of LREC 2018*.

Yu-An Chung et al. 2018. Unsupervised cross-modal alignment of speech and text embedding spaces. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Alexis Conneau et al. 2020. Unsupervised cross-lingual representation learning for speech recognition. *arXiv:2006.13979*.

J. Cui et al. 2015. Multilingual representations for low resource speech recognition and keyword search. In *ASRU 2015*, pages 259–266.

Aliya Deri and Kevin Knight. 2016. Grapheme-to-phoneme models for (almost) any language. In *Proceedings of ACL 2016*, pages 399–408, Berlin, Germany. ACL.

Moussa Doumbouya, Lisa Einstein, and Chris Piech. 2021. Using radio archives for low-resource speech recognition: Towards an intelligent virtual assistant for illiterate users. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35.

Siyuan Feng et al. 2021. How phonotactics affect multilingual and zero-shot ASR performance. In *ICASSP 2021*.

Ben Foley et al. 2018. Building speech recognition systems for language documentation: The CoEDL endangered language pipeline and inference system. In *Proceedings of SLTU 2018*.

Jason Fong, Jason Taylor, Korin Richmond, and Simon King. 2019. A comparison of letters and phones as input to sequence-to-sequence models for speech synthesis. In *Proceedings of ISCA SSW 2019*, pages 223–227.

Kyle Gorman and Richard Sproat. 2016. Minimally supervised number normalization. *Transactions of the Association for Computational Linguistics*, 4:507–519.

Anmol Gulati et al. 2020. Conformer: Convolution-augmented transformer for speech recognition. In *Proceedings of Interspeech 2020*, pages 5036–5040.

Vishwa Gupta and Gilles Boulianne. 2020. Automatic transcription challenges for Inuktitut, a low-resource polysynthetic language. In *Proceedings of LREC 2020*, pages 2521–2527, Marseille, France. ELRA.

Mark Hasegawa-Johnson, Camille Goudeseune, and Gina-Anne Levow. 2019. Fast transcription of speech in low-resource languages. *arXiv:1909.07285*.

Mark Hasegawa-Johnson et al. 2020. Grapheme-to-phoneme transduction for cross-language ASR. In *Statistical Language and Speech Processing*, pages 3–19, Cham. Springer International Publishing.

Tomoki Hayashi et al. 2020. Espnet-TTS: Unified, reproducible, and integratable open source end-to-end text-to-speech toolkit. In *ICASSP 2020*, pages 7654–7658. IEEE.

Mutian He, Jingzhou Yang, and Lei He. 2021. Multilingual byte2speech text-to-speech models are few-shot spoken language learners. *arXiv:2103.03541*.

Stephanie Hirmer et al. 2021. Building representative corpora from illiterate communities: A review of challenges and mitigation strategies for developing countries. In *Proceedings of EACL 2021*.

Nils Hjortnaes et al. 2020. Improving the language model for low-resource ASR with online text corpora. In *Proceedings of SLTU-CCURL 2020*, pages 336–341, Marseille, France. ELRA.

Mathieu Hu et al. 2020. Kaldi-web: An installation-free, on-device speech recognition system. In *Proceedings of Interspeech 2020: Show & Tell*, Shanghai, China.

Thad Hughes et al. 2010. Building transcribed speech corpora quickly and cheaply for many languages. In *Proceedings of Interspeech 2010*, pages 1914–1917.

Pratik Joshi et al. 2020. The state and fate of linguistic diversity and inclusion in the NLP world. In *Proceedings of ACL 2020*, pages 6282–6293, Online. ACL.

Anjuli Kannan et al. 2019. Large-scale multilingual speech recognition with a streaming end-to-end model. In *Proceedings of Interspeech 2019*, pages 2130–2134.

Kyle Kastner, João Felipe Santos, Yoshua Bengio, and Aaron Courville. 2019. Representation mixing for TTS synthesis. In *Proceedings of ICASSP 2019*.

Leonard Katz and Ram Frost. 1992. The reading process is different for different orthographies: The orthographic depth hypothesis. In Leonard Katz and Ram Frost, editors, *Orthography, Phonology, Morphology, and Meaning*, page 67–84. Elsevier North Holland Press, Amsterdam.

Chanwoo Kim, Dhananjaya Gowda, Dongsoo Lee, Jiyeon Kim, Ankur Kumar, Sungsoo Kim, Abhinav Garg, and Changwoo Han. 2020. A review of on-device fully neural end-to-end automatic speech recognition algorithms. *arXiv:2012.07974*.

Young-Bum Kim and Benjamin Snyder. 2013. Optimal data set selection: An application to grapheme-to-phoneme conversion. In *Proceedings of NAACL 2013*, pages 1196–1205, Atlanta, Georgia. ACL.

S. H. Krishnan Parthasarathi and N. Strom. 2019. Lessons from building acoustic models with a million hours of speech. In *ICASSP 2019*, pages 6670–6674.

Stephen Laurence and Eric Margolis. 2001. The poverty of the stimulus argument. *The British Journal for the Philosophy of Science*, 52(2):217–276.

Jackson L. Lee et al. 2020. Massively multilingual pronunciation modeling with WikiPron. In *Proceedings of LREC 2020*, pages 4223–4228.

B. Li, Y. Zhang, T. N. Sainath, Y. Wu, and W. Chan. 2019. Bytes are all you need: End-to-end multilingual speech recognition and synthesis with bytes. In *Proceedings of ICASSP 2019*.

Xinjian Li et al. 2020. Universal phone recognition with a multilingual allophone system. In *ICASSP 2020*, pages 8249–8253. IEEE.

Yusen Lin, Jiayong Lin, Shuaicheng Zhang, and Haoying Dai. 2021. Bilingual dictionary-based language model pretraining for neural machine translation.

Chunxi Liu et al. 2020. Multilingual graphemic hybrid ASR with massive data augmentation. In *Proceedings of SLTU-CCURL 2020*, pages 46–52, Marseille, France. ELRA.

M. Mohri, F. Pereira, and M. Riley. 2002. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88.

Steven Moran and Daniel McCloy, editors. 2019. *PHOIBLE 2.0*. Max Planck Institute for the Science of Human History, Jena.

David R. Mortensen, Siddharth Dalmia, and Patrick Littell. 2018. Epitran: Precision G2P for many languages. In *Proceedings of LREC 2018*, Miyazaki, Japan. ELRA.

A. Oktem et al. 2020. Gamayun - language technology for humanitarian response. In *2020 IEEE Global Humanitarian Technology Conference*, pages 1–4.

Pedro Javier Ortiz Suárez, Laurent Romary, and Benoît Sagot. 2020. A monolingual approach to contextualized word embeddings for mid-resource languages. In *Proceedings of ACL 2020*, pages 1703–1714, Online. ACL.

V. Panayotov et al. 2015. Librispeech: An ASR corpus based on public domain audio books. In *ICASSP 2015*, pages 5206–5210.

Daniel S. Park et al. 2019. SpecAugment: A simple data augmentation method for automatic speech recognition. *Proceedings of Interspeech 2019*, pages 2613–2617.

Daniel S. Park et al. 2020. SpecAugment on large scale datasets. *ICASSP 2020*, pages 6879–6883.

Niko Partanen and Michael Rießler. 2019. An OCR system for the Unified Northern Alphabet. In *Proceedings of the Fifth Workshop on Computational Linguistics for Uralic Languages*, pages 77–89, United States. ACL.

Matthias Petursson, Simon Klüpfel, and Jon Gudnason. 2016. Eyra - speech data acquisition system for many languages. In *Proceedings of SLTU 2016*, Yogyakarta, Indonesia.

Cal Peyser et al. 2019. Improving performance of end-to-end ASR on numeric sequences. In *Proceedings of Interspeech 2019*.

Jonas Pfeiffer et al. 2020. AdapterHub: A framework for adapting transformers. In *Proceedings of EMNLP 2020: Systems Demonstrations*, pages 46–54, Online. ACL.

Daniel Povey et al. 2011. The Kaldi speech recognition toolkit. In *ASRU 2011*.

Rohit Prabhavalkar, Kanishka Rao, Tara N. Sainath, Bo Li, Leif Johnson, and Navdeep Jaitly. 2017. A comparison of sequence-to-sequence models for speech recognition. In *Proceedings of Interspeech 2017*, pages 939–943.

Manasa Prasad, Theresa Breiner, and Daan van Esch. 2018. Mining training data for language modeling across the world's languages. In *Proceedings of SLTU 2018*.

Manasa Prasad et al. 2019. Building large-vocabulary asr systems for languages without any audio training data. In *Proceedings of Interspeech 2019*.

Vineel Pratap et al. 2020. Massively Multilingual ASR: 50 Languages, 1 Model, 1 Billion Parameters. In *Proceedings of Interspeech 2020*, pages 4751–4755.

S. Punjabi, H. Arsikere, and S. Garimella. 2019. Language model bootstrapping using neural machine translation for conversational speech recognition. In *ASRU 2019*, pages 487–493.

Shruti Rijhwani, Antonios Anastasopoulos, and Graham Neubig. 2020. OCR post correction for endangered language texts. In *Proceedings of EMNLP 2020*, pages 5931–5942, Online. Association for Computational Linguistics.

Sandy Ritchie et al. 2019. Unified verbalization for speech recognition synthesis across languages. In *Proceedings of Interspeech 2019*.

Sandy Ritchie et al. 2020. Data-driven parametric text normalization: Rapidly scaling finite-state transduction verbalizers to new languages. In *Proceedings of SLTU-CCURL 2020*.

Andrew Rosenberg et al. 2019. Speech recognition with augmented synthesized speech. In *ASRU 2019*.

Attapol Rutherford, Fuchun Peng, and Françoise Beaufays. 2014. Pronunciation learning for named-entities through crowd-sourcing. In *Proceedings of Interspeech 2014*.

Tara N. Sainath, Rohit Prabhavalkar, Shankar Kumar, Seungji Lee, Anjuli Kannan, David Rybach, Vlad Schogol, Patrick Nguyen, Bo Li, Yonghui Wu, Zhifeng Chen, and Chung-Cheng Chiu. 2018. No need for a lexicon? Evaluating the value of the pronunciation lexica in end-to-end models. In *Proceedings of ICASSP 2018*.

H. Sak et al. 2013. Language model verbalization for automatic speech recognition. In *ICASSP 2013*.

O. Scharenborg et al. 2020. Speech technology for unwritten languages. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:964–975.

S. Schneider et al. 2019. wav2vec: Unsupervised pre-training for speech recognition. *Proceedings of Interspeech*, pages 3465–3469.

Tanja Schultz and Alex Waibel. 2001. Experiments on cross-language acoustic modeling. In *Proceedings of the 7th European Conference on Speech Communication and Technology*.

Frank Seifart et al. 2018. Language documentation twenty-five years on. *Language*, 94(4):e324–e345.

Jack Serrino, Leonid Velikovich, Petar Aleksic, and Cyril Allauzen. 2019. Contextual recovery of out-of-lattice named entities in automatic speech recognition. In *Proceedings of Interspeech 2019*, pages 3830–3834, Graz, Austria.

Jiatong Shi, Jonathan D. Amith, Rey Castillo García, Esteban Guadalupe Sierra, Kevin Duh, and Shinji Watanabe. 2021. Leveraging end-to-end asr for endangered language documentation: An empirical study on yoloxóchitl mixtec.

RJ Skerry-Ryan, Eric Battenberg, Ying Xiao, Yuxuan Wang, Daisy Stanton, Joel Shor, Ron J. Weiss, Rob Clark, and Rif A. Saurous. 2018. Towards end-to-end prosody transfer for expressive speech synthesis with Tacotron. In *Proceedings of ICML 2018*.

Jose Sotelo, Soroush Mehri, Kundan Kumar, João Felipe Santos, Kyle Kastner, Aaron Courville, and Yoshua Bengio. 2017. Char2Wav: End-to-end speech synthesis. In *Proceedings of ICLR 2017*.

Richard Sproat, Alan W. Black, Stanley Chen, Shankar Kumar, Mari Ostendorf, and Christopher Richards. 2001. Normalization of non-standard words. *Computer Speech and Language*, 15(3):287–333.

Richard Sproat and Keith Hall. 2014. Applications of maximum entropy rankers to problems in spoken language processing. In *Proceedings of Interspeech 2014*.

Richard Sproat and Navdeep Jaitly. 2017. RNN approaches to text normalization: A challenge. *arXiv:1611.00068*.

Piotr Szymański et al. 2020. WER we are and WER we think we are. In *Findings of EMNLP 2020*, pages 3290–3295, Online. ACL.

Jason Taylor and Korin Richmond. 2019. Analysis of Pronunciation Learning in End-to-End Speech Synthesis. In *Proceedings of Interspeech 2019*, pages 2070–2074.

Anjana Vakil et al. 2014. lex4all: A language-independent tool for building and evaluating pronunciation lexicons for small-vocabulary speech recognition. In *Proceedings of ACL 2014: System Demonstrations*, pages 109–114, Baltimore, Maryland. ACL.

A.P.J. van den Bosch, A. Content, W.M.P. Daelemans, and B.L.M.F. de Gelder. 1994. Measuring the complexity of writing systems. *Journal of Quantitative Linguistics*, 1(3):178–188. Pagination: 11.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv:1807.03748*.

Daan van Esch, Mason Chua, and Kanishka Rao. 2016. Predicting pronunciations with syllabification and stress with recurrent neural networks. In *Proceedings of Interspeech 2016*.

Daan van Esch and Richard Sproat. 2017. An expanded taxonomy of semiotic classes for text normalization. In *Proceedings of Interspeech 2017*.

Daan van Esch et al. 2019. Writing across the world's languages: Deep internationalization for Gboard, the Google keyboard. Technical report.

Nanne van Noord et al. 2021. Automatic annotations and enrichments for audiovisual archives. In *ICAART 2021*.

Ehsan Variani, David Rybach, Cyril Allauzen, and Michael Riley. 2020. Hybrid autoregressive transducer (HAT). In *Proceedings of ICASSP 2020*.

Shafqat Mumtaz Virk et al. 2020. The DReaM corpus: A multilingual annotated corpus of grammars for the world's languages. In *Proceedings of LREC 2020*, pages 878–884, Marseille, France. ELRA.

Yuxuan Wang, R.J. Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc Le, Yannis Agiomyrgiannakis, Rob Clark, and Rif A. Saurous. 2017. Tacotron: Towards end-to-end speech synthesis. In *Proceedings of Interspeech 2017*, pages 4006–4010.

Shinji Watanabe et al. 2018. ESPnet: End-to-end speech processing toolkit. In *Proceedings of Interspeech 2018*, pages 2207–2211.

Guillaume Wisniewski, Séverine Guillaume, and Alexis Michaud. 2020. Phonemic transcription of low-resource languages: To what extent can preprocessing be automated? In *Proceedings of SLTU-CCURL 2020*, pages 306–315, Marseille, France. ELRA.

Yusuke Yasuda, Xin Wang, and Junichi Yamagishi. 2020. Investigation of learning abilities on linguistic features in sequence-to-sequence text-to-speech synthesis. *arXiv:2005.10390*.

Piotr Zelasko et al. 2020. That Sounds Familiar: An Analysis of Phonetic Representations Transfer Across Languages. In *Proceedings of Interspeech 2020*, pages 3705–3709.

Hao Zhang, Richard Sproat, Axel H. Ng, Felix Stahlberg, Xiaochang Peng, Kyle Gorman, and Brian Roark. 2019. Neural models of text normalization for speech applications. *Comput. Linguist.*, 45(2):293–337.

Towards Fast Cognate Alignment on Imbalanced Data

Logan Born[1], **M. Willis Monroe**[2], **Kathryn Kelley**[3], **Anoop Sarkar**[1]
[1]Simon Fraser University, [2]University of New Brunswick, [3]Università di Bologna
loborn@sfu.ca, willis.monroe@unb.ca, kathrynerin.kelley@unibo.it, anoop@cs.sfu.ca

Abstract
Cognate alignment models purport to enable decipherment, but their speed and need for clean data can make them unsuitable for realistic decipherment problems. We seek to draw attention to these shortcomings in the hopes that future work may avoid them, and we outline two techniques which begin to overcome the described problems.

Keywords: cognate detection, alignment, decipherment

1. Introduction

Cognate alignment models aim to identify cognate lexeme pairs by aligning word lists from related languages; when one of these lists comes from an undeciphered language, this alignment produces a decipherment for any words which are correctly mapped to their cognates. In this study, we highlight limitations of current cognate alignment models which restrict their usefulness in practical decipherment tasks, and propose partial solutions for more realistic data.

2. Motivation

The state-of-the art for cognate alignment (Tamburini, 2023) uses coupled simulated annealing (Xavier-de-Souza et al., 2010) to solve a search problem over the set of all k-permutations mapping n lost-language words into k known-language buckets. The prior state-of-the-art Luo et al. 2019 alternated between learning character- and word-level pairings using iterative expectation-maximization-style training. Both achieve strong results when every word has at least one cognate in the language it has been paired with; however, this is artificially clean compared to true decipherment, where not only are unpaired words likely, but there may also be uncertainties about the underlying word boundaries or character inventory. Luo et al. (2019) acknowledge this and evaluate on less-clean Ugaritic-Old Hebrew data, in which setting their accuracy drops to just 65.9%, from 93.5% on clean data. Tamburini 2023 does not include noisy evaluations, likely because their search is computationally intensive and does not appear scalable to settings with many unpaired words.

Luo et al. (2019, 3152) also "found it beneficial to train [...] only on a randomly selected subset (10%) of the entire corpus with the same percentage of noncognates," but observe that such filtering is impossible in a realistic setting where it is not known which words *have* cognates. On undeciphered data, both word lists would need to be sampled independently, meaning that a 10% subset of the corpus should be expected to contain a mere 1% of the original cognate pairs, destroying most of the signal that the model would learn from.

Finally, note that existing models can be inefficient even on clean data (Tamburini 2023, 89 "took a relevant time to converge"), which is undesirable for real decipherments where the correct target language is not known beforehand. Such settings may require aligning to multiple targets before a true solution can be found, and this scattershot approach is only feasible when each individual language pair can be aligned efficiently.

3. LMs for Character-Level Alignment

Tran (2020) considers the task of adapting pretrained models to new languages. Given a matrix of pretrained word embeddings $\mathbf{E}_e$, they propose to derive embeddings $\mathbf{E}_f$ for a new language using linear combinations of the rows of $\mathbf{E}_e$:

$$\mathbf{E}_f[i] = \sum_{j=1}^{|V_e|} \alpha_{ij}\mathbf{E}_e[j] = \alpha_i \mathbf{E}_e \qquad (1)$$

where α_i is a sparse weight vector satisfying $\sum_j^{|V_e|} \alpha_{ij} = 1$. The authors cite two offline approaches (Dyer et al., 2013; Conneau et al., 2017) which can be used to estimate α by exploiting sentence-level context. If the dependence on sentence-level context (which is absent from the word lists used for cognate alignment) could be eliminated, we suggest that a similar technique could by applied to the embeddings from a *character*-level language model to quickly learn character equivalencies between two scripts or languages as a first step towards cognate alignment.

Concretely, we propose to first train a character-level language model on the known language. The inputs to the model are individual words from the known-language word list, with no additional context, tokenized at the character level. Given the

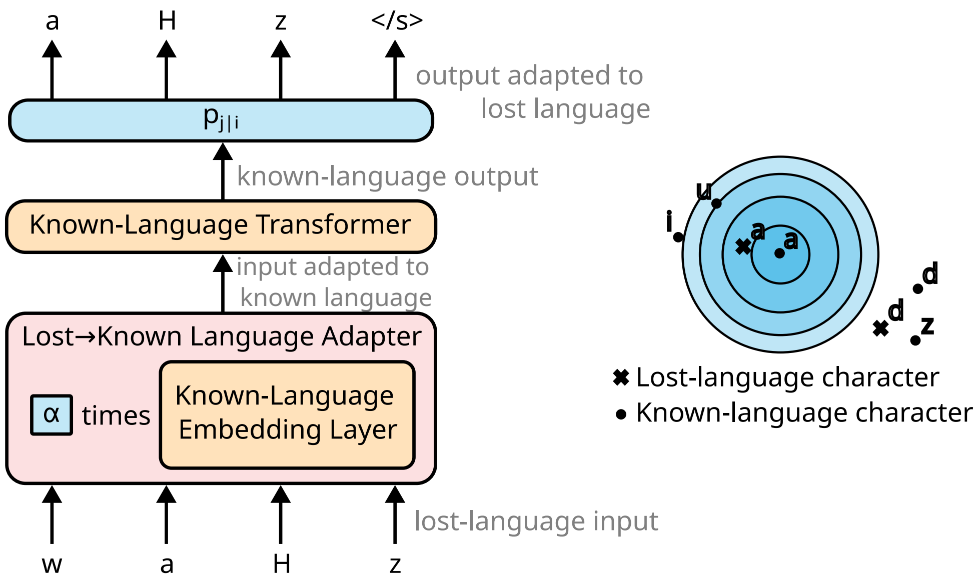
a
H
z
</s>
output adapted to lost language
p_{j|i}
known-language output
Known-Language Transformer
input adapted to known language
Lost→Known Language Adapter
α times
Known-Language Embedding Layer
lost-language input
w
a
H
z
u
i
a
a
d
d
z
✖ Lost-language character
● Known-language character

Figure 1: Schematic view of proposed architecture (left) and sampling procedure for Equation 2 (right). Characters are embedded using a matrix E (for the known language) or αE (for the lost language). The probability that character a in the known-language corresponds to character a in the lost language is proportional to their distance in this embedding space. During fine-tuning, α (and by extension $p_{j|i}$, which depends on α) is the only parameter that is allowed to be updated.

small size of cognate detection datasets, we use a shallow Transformer (Vaswani et al., 2017) with a small feature dimension (2 layers, 4 heads, and 32 dimensions); this is the largest model which we are able to train reliably, as deeper models or those with larger dimension often fail to converge. We use positional encodings (Vaswani et al., 2017) and apply dropout at a rate of 0.5. This model is trained with SGD to minimize categorical cross-entropy on an autoregressive language modeling task using a causal attention mask.

Let $E \in \mathbb{R}^{k\times 32}$ be the embedding layer of this model, where k is the number of known-language characters. Let $M : \mathbb{R}^{n\times 32} \rightarrow \mathbb{R}^{n\times k}$ be a black-box representation for the remainder of the model, which maps a sequence of n 32-dimensional character embeddings onto a sequence of log-probabilities over k known characters.

We next introduce a mapping $\alpha \in \mathbb{R}^{l\times k}$ following Tran 2020. The product $\alpha E \in \mathbb{R}^{l\times 32}$ can be seen as an embedding matrix for l distinct lost-language characters, and $M\circ\alpha E$ can be seen as a hybrid language model which accepts lost-language inputs and predicts known-language outputs. To convert the outputs from $M \circ \alpha E$ into a distribution over lost-language characters, we introduce the following conditional probability distribution inspired by t-SNE (Hinton and Roweis, 2002; van der Maaten and Hinton, 2008):

$$\log p_{j|i} = \frac{-|\mathbf{x}_i - \alpha E_j|^2/2\sigma_i}{\sum_{h\neq i} -|\mathbf{x}_i - \alpha E_h|^2/2\sigma_i} \qquad (2)$$

where $1 \leq i \leq k$, $1 \leq j \leq l$, $\mathbf{x}_i$ is the embedding for the ith known character, αE_j is the embedding for the jth lost character, and σ_i is a per-character density estimate. Given a known-language character i, suppose we sample neighboring characters in the embedding space based on their distance from $\mathbf{x}_i$, with Gaussian falloff. Assuming we are only allowed to sample neighbors from the lost language, $p_{j|i}$ models the probability that the lost-language character j will be the one sampled. This is equivalent to the sampling procedure used in t-SNE (van der Maaten and Hinton, 2008) with the modification that points are divided into two classes (known and lost), and each class can only sample points from the other class.

Given a distribution $\mathbf{p} = [p_1, ..., p_k]$ over known-language characters returned as output by $M \circ \alpha E$, we model the corresponding distribution $\tilde{\mathbf{p}} = [\tilde{p}_1, ..., \tilde{p}_l]$ over *lost*-language characters as $\tilde{p}_j = \sum_{i=1}^{k} p_i p_{j|i}$. With this transformation in hand, we have adapted the original known-language model to both accept lost-language inputs *and* predict lost-language outputs using only the mapping α.

We now propose to fine-tune the adapted model on the *lost*-language word list, following the same procedure used to train the underlying known-language model. However, we make α the only tunable parameter, so that the only way for the model to improve the language modeling loss at this stage will be to learn the correct mappings between characters in the two scripts. This procedure makes α *trainable*, whereas the mapping in Tran 2020 was static and estimated offline.

Permutation Loss We hypothesize that α may be more easily learned if it is constrained to be approximately one-to-one, as mappings between scripts will generally be sparse. Thus we generalize the matrix penalty function from Lyu et al. (2020) to the case of non-square $k \times l$ matrices:

$$\mathcal{L}_{sparse} = \sum_{i=1}^{k} \left[\sum_{j=1}^{l} |\alpha_{ij}| - \left(\sum_{j=1}^{l} \alpha_{ij}^2 \right)^{1/2} \right] \tag{3}$$
$$+ \sum_{j=1}^{l} \left[\sum_{i=1}^{k} |\alpha_{ij}| - \left(\sum_{i=1}^{k} \alpha_{ij}^2 \right)^{1/2} \right]$$

When applied to a square matrix α, this quantity approaches zero as the matrix approaches a permutation; in the non-square setting it approaches zero as α approaches some non-square projection of a permutation. We hypothesize that tuning α to jointly minimize the sum of the cross-entropy language modeling loss with this sparsity loss will yield more accurate character-level alignments than fine-tuning on the language modeling loss by itself.

Note that, during our fine-tuning step, there is no way for a character in one script to be decoded to multiple characters in the other script; we use a vanilla Transformer architecture with no mechanism for explicit insertion or deletion operations. This creates an inductive bias towards one-to-one mappings, which is reinforced by this sparsity loss term. Despite this, we will show in Section 5.2 that our model is nonetheless capable of learning more complex, many-to-one mappings when there is evidence for such in the input data.

4. Towards Word-Level Alignment

After fine-tuning, α yields a representation for each lost-language character as a linear combination of known-language characters. We next wish to use these combinations to infer likely pairings between cognates at the word level.

4.1. Edit Distances

We first compute the Levenshtein distance (Levenshtein, 1966) from every lost word to every known word, where the cost of substituting lost character j with known character i is:

$$C(j, i) = 1 - \frac{p_{i|j}}{\max_i p_{i|j}} \tag{4}$$

where $p_{i|j}$ is the conditional probability that known character i would be sampled by lost character j paralleling Eq. (2). Note that $C(j, i) = 0$ whenever i is the nearest known-language neighbor to j in the embedding space, while for all other pairings the cost rises proportionally to the distance between i and j. Thus there is no cost to replace a lost-language character with its most likely known-language correspondent, and increasing cost for less likely substitutions.

4.2. Alignment Likelihoods

For lost- and known-language vocabularies V_l and V_k, let $\Delta \in \mathbb{R}^{|V_l| \times |V_k|}$ be a matrix where Δ_{mn} is the weighted edit distance between lost word m and known word n as described above. Let $D_m = \text{softmax}(-\Delta_m)$ be a probability distribution where D_{mn} is the probability that lost word m aligns to known word n, and note that the resulting probabilities are inversely proportional to the original edit distances (Eq. (4)).

To obtain a more sophisticated model for word-level alignments, we can incorporate a prior estimate for the likelihood that known word n is cognate to some lost word, as opposed to being a distractor that has no mapping into the other language. Existing approaches to cognate detection excel in clean settings where there are few or no such distractors, so the ability to identify and prune these words would be a useful result in itself.

To this end, we propose to learn a language model on the *lost* word list, then fine-tune on the known word list following the same procedure outlined in Section 3. In the resulting model, the average perplexity when producing known word n should be low if n is cognate to some lost word, as in this case the underlying lost-language model will have seen that cognate during pretraining and the corresponding known word will look “in-domain”. By contrast, if known word n is not cognate to any lost words, it should be “out-of-domain” and therefore incur a higher average perplexity. We construct a vector $\Psi \in \mathbb{R}^k$ where Ψ_n is the average perplexity when producing the characters in known word n. We convert this to a probability distribution $P = \text{softmax}(-\Psi)$, and estimate the probability that lost word m aligns to known word n as $P_n^r D_{mn}$ where r is a smoothing term.

5. Experimental Results

5.1. Character Alignment

We train the proposed model on the noisy Ugaritic-Old Hebrew data from Luo et al. 2019. We consider two directions: pre-training on the known language and fine-tuning α on the lost language, and *vice versa*. In each direction, we compare models trained (i) without the permutation loss $\mathcal{L}_{sparse}$, (ii) with the permutation loss for just the first 50 iterations as a kind of warm-up, and (iii) with the permutation loss for the entire duration of training.

	Permutation Loss					
	None		**Warm-Up**		**Always**	
Pretraining Language	**top 1**	**top 5**	**top 1**	**top 5**	**top 1**	**top 5**
Old Hebrew (Known)	26%	57%	**83%**	**100%**	13%	30%
Ugaritic (Lost)	48%	74%	48%	70%	0%	17%

Table 1: Top-1 and top-5 precision of character-level mappings derived from α.

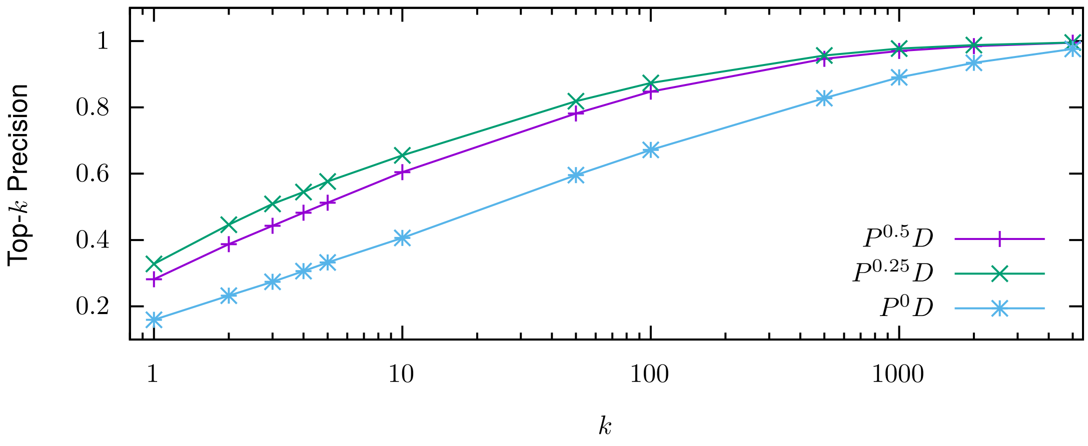

Figure 2: Top-k precision on Ugaritic-Old Hebrew cognate detection for various thresholds k and for values of the smoothing parameter $r \in [0, 0.25, 0.5]$.

α captures a nuanced mapping between characters in the two scripts in the form of a weighted sum. We attempt to concretize this into a one-to-one mapping for evaluation purposes, but note that this will necessarily lose some of the information inherent in the full set of weights. For example, Table 1 reports top-1 and top-5 precision for mappings derived by aligning each character in the known script to the character(s) with the maximmum likelihood $p_{j|i}$ according to Eq. (2).

From these results it is clear that the proposed technique can accurately learn cross-script character equivalencies, doing so most effectively when pretrained on the *known* language and fine-tuned on the lost. In this setting, α appears to perfectly capture the mapping between the two scripts, achieving 100% top-5 precision in the best case. In the opposite direction, the best result is just 74% top-5 precision. We speculate that this asymmetry derives from the fact that there is much more known- than lost-language data, meaning that the initial model will learn higher-quality character representations when pre-trained on the known language.

Our proposed permutation loss also appears to improve the character-level alignment under certain training regimes. Specifically, when pre-training on the known language, applying the permutation loss during the earliest iterations of the alignment step yields significant improvements in the quality of the learned alignment. In other settings, this term has no effect or is actively detrimental to the quality of the learned mapping. This suggests that it can be useful to “prime” the model to expect a roughly one-to-one mapping at the start of training, but that this requirement must eventually be relaxed. This makes sense given that the true mapping between these scripts includes some many-to-one or one-to-many relationships (as between Ugaritic *a, u, i* and Old Hebrew *a* in most contexts).

5.2. Word Alignment

The data contains 38898 total Hebrew words; thus *a priori* each lost (Ugaritic) word could be aligned to any of 38898 possible targets. We wish to narrow this to a small pool of candidate cognates for each word: ideally, we want just the most likely cognate in each case, but even tens of candidates per word is manageable for reranking or human evaluation.

To this end, for each lost word, we select the k known words with the highest probability according to the distribution P^rD described above. We call these *candidate cognate pairs*, and consider the alignment successful if the true cognate is among the k chosen terms. Figure 2 plots the top-k precision for different values of k and the smoothing parameter r.

Our best results arrive at $r = 0.25$, where nearly half (46%) of true cognates are either the most likely or the second-most likely candidate according to P^rD, and 66% fall within the top 10 most likely candidates. 66% is the same accuracy reported by Luo et al. (2019) for their noisy evaluation. These results suggest an avenue for future work whereby the top-k candidates are reranked to promote the *true* cognates to the top of the ranking. Under this approach, the correct cognate would only need to be selected from a few tens of candidates, rather than the full set of many thousands.

For additional context on these results, note that half of the word pairs in the input data have identical lengths in both languages, while the other half are longer in one language than the other. Thus any score above 50% must include some cases where the model correctly infers a complex, non-one-to-one mapping between some characters, such as the deletion of `y` from Heb `labyk` to Uga `labk` or of `awy` from Heb `wlawyb` to Uga `wlib`.

We emphasize that top-k precision is highest when we exploit P as a prior estimate for whether a word has a cognate or not. Thus P provides a usable signal to help assess whether or not a given word is a distractor. This is encouraging, as it is otherwise difficult to tell which words are "out-of-domain" when dealing with undeciphered data.

The process outlined in this work is also extremely fast, averaging just 0.015s per word, versus 0.464s per word for Luo et al. 2019 on the same data and longer still for Tamburini 2023. We are therefore optimistic that this work can serve as the basis for faster, more efficient cognate alignment models. One straightforward application of these results would be as a smart initialization for a model such as Luo et al. 2019, to bias the initial character mappings and limit the set of word-level pairings that are explored. This would increase the efficiency of such a model by constraining its search space, which we predict would also help to limit the impact of distractor vocabulary items.

6. Related Work

Aside from the works noted in Section 2, cognate prediction (Fourrier and Sagot, 2022; Fourrier et al., 2021; Hämäläinen and Rueter, 2019; Wu and Yarowsky, 2018; Dekker, 2018) is a closely-related task found in the field of historical linguistics, whereby the form of a word's cognate in some target language is predicted given that word's phonetic representation in some known language. Cognate prediction models are typically generative, producing a sequence of output phonemes given a sequence of input phonemes. Thus these models have an open output vocabulary (the set of all phoneme sequences), and their inputs and outputs come from the same script (IPA or another phonetic representation). By contrast, our model selects candidate cognates from a closed list (similarly to Beinborn et al. 2013), and assumes that inputs and outputs are written using distinct scripts. This enables the application of our model to undeciphered data, where the underlying phonetic representations are unknown, but also limits it to finding only those word pairs which are attested in the input word lists. Moreover, cognate prediction requires parallel training data, whereas our decipherment-focused approach learns from non-aligned word lists. Notably, if we omit the t-SNE inspired transformation which was used to convert model outputs from one script to the other, and omit the following edit-distance computations, our model would function as a transducer which rewrites inputs in one script into another script, in a way that would more closely resemble prior cognate prediction models. It may therefore be worth exploring applications of our model to this task in future work.

7. Conclusion

In conclusion, our work highlights the need for cognate alignment models to be both efficient and robust against noise if they are to be applied to realistic decipherment tasks. We argue that these qualities are lacking in existing approaches. As a first step towards overcoming these challenges, we have introduced a novel technique that leverages monolingual language models to swiftly and accurately learn cross-script character equivalencies. By incorporating a permutation loss term, we further improve the precision of the learned equivalencies by guiding the model towards nearly one-to-one mappings. Finally, we propose a method of combining weighted edit distances with perplexity signals which for the first time enables effective filtering of words without cognates in the paired language. These advancements lay the groundwork for the development of more efficient and reliable cognate alignment models in future work.

8. Bibliographical References

Lisa Beinborn, Torsten Zesch, and Iryna Gurevych. 2013. Cognate production using character-based machine translation. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 883–891, Nagoya, Japan. Asian Federation of Natural Language Processing.

Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou.

2017. Word translation without parallel data. *arXiv preprint arXiv:1710.04087*.

Peter Dekker. 2018. Msc thesis: Reconstructing language ancestry by performing word prediction with neural networks.

Chris Dyer, Victor Chahuneau, and Noah A. Smith. 2013. A simple, fast, and effective reparameterization of IBM model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648, Atlanta, Georgia. Association for Computational Linguistics.

Clémentine Fourrier, Rachel Bawden, and Benoît Sagot. 2021. Can cognate prediction be modelled as a low-resource machine translation task? In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 847–861, Online. Association for Computational Linguistics.

Clémentine Fourrier and Benoît Sagot. 2022. Probing multilingual cognate prediction models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 3786–3801, Dublin, Ireland. Association for Computational Linguistics.

Mika Hämäläinen and Jack Rueter. 2019. Finding Sami cognates with a character-based NMT approach. In *Proceedings of the 3rd Workshop on the Use of Computational Methods in the Study of Endangered Languages Volume 1 (Papers)*, pages 39–45, Honolulu. Association for Computational Linguistics.

Geoffrey E Hinton and Sam Roweis. 2002. Stochastic neighbor embedding. In *Advances in Neural Information Processing Systems*, volume 15. MIT Press.

V. I. Levenshtein. 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707.

Jiaming Luo, Yuan Cao, and Regina Barzilay. 2019. Neural decipherment via minimum-cost flow: From Ugaritic to Linear B. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3146–3155, Florence, Italy. Association for Computational Linguistics.

Jiancheng Lyu, Shuai Zhang, Yingyong Qi, and Jack Xin. 2020. Autoshufflenet: Learning permutation matrices via an exact lipschitz continuous penalty in deep convolutional neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 608–616, New York, NY, USA. Association for Computing Machinery.

Fabio Tamburini. 2023. Decipherment of lost ancient scripts as combinatorial optimisation using coupled simulated annealing. In *Proceedings of the Workshop on Computation and Written Language (CAWL 2023)*, pages 82–91, Toronto, Canada. Association for Computational Linguistics.

Ke Tran. 2020. From english to foreign languages: Transferring pre-trained language models.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Winston Wu and David Yarowsky. 2018. Creating large-scale multilingual cognate tables. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Samuel Xavier-de-Souza, Johan A. Suykens, Joos Vandewalle, and Désiré Bolle. 2010. Coupled simulated annealing. *IEEE Trans Syst Man Cybern B Cybern*, 40(2):320–335.

Simplified Chinese Character Distance Based on Ideographic Description Sequences

Yixia Wang and Emmanuel Keuleers
Tilburg University
Warandelaan 2, 5037 AB Tilburg
y.wang_1@tilburguniversity.edu, e.a.keuleers@tilburguniversity.edu

Abstract

Character encoding systems have long overlooked the internal structure of characters. Ideographic Description Sequences, which explicitly represent spatial relations between character components, are a potential solution to this problem. In this paper, we illustrate the utility of Ideographic Description Sequences in computing edit distance and finding orthographic neighbors for Simplified Chinese characters. In addition, we explore the possibility of using Ideographic Description Sequences to encode spatial relations between components in other scripts.

Keywords: Ideographic Description Sequences, Character distance, Character Neighbors

1. Introduction

Storage and communication of written text using digital computers requires conventions for encoding characters. Early efforts at establishing encoding standards were driven by practicality and economy of space. Developed in 1963, the American Standard Code for Information Interchange (ASCII; American National Standards Institute, 1995) lies at the basis of most character encoding systems in use today. ASCII uses a 7-bit encoding, with 32 of the 128 positions allocated to communication control characters and the other 96 reserved for numbers, upper- and lowercase letters of the English alphabet, and punctuation. As computer technology spread, ASCII was succeeded by ISO-8859 (ISO/IEC, 1987) which, with 8-bit encoding and language specific versions, enabled the encoding of characters for a wider variety of alphabetic writing systems (e.g., ISO 8859-5 for Cyrillic, ISO-8859-11 for Thai). Accommodation for storing the many characters used in CJK (Chinese, Japanese, Korean) writing systems, came with the Unicode Standard, with different variants allowing for up-to 32-bit encoding (>4 billion characters).

The legacy of ASCII lead to these successive standards allocating more and more space for individual characters instead of incorporating *compositionality*, which is a design feature of most writing systems (English writing being a notable exception). For instance, for writers of French it is understood that most vowels can be accented, yet ISO-8859-1 has different slots for â, ê, î, ô, and û; á, é, í, ó, and ú; etc. For other writing systems, such as Chinese, compositionality is the norm, rather than the exception.

Recognizing that it was necessary to represent characters that do not have a dedicated slot, such as rare or novel Chinese characters, Unicode 15.1 (Unicode Consortium, 2023) introduced Ideographic Description Sequences (IDSs) as a principled approach to encoding characters compositionally.[1] Figure 1 shows the rendition of a rare character using an IDS.

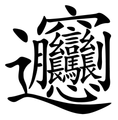
Figure 1: Chinese character *biang1* 'the sound of slapping and kneading noodles during noodle-making'. The ideographic description sequence for this character is ⿺辶⿱穴⿲月⿱⿲幺言幺⿲長馬長 刂心.

Because of their ability to encode and represent characters compositionally, IDSs have a wide range of applications. In this paper, we will focus on a novel application, namely the use of IDSs to compute distance between Chinese characters. As section 2 will show, psycholinguistic literature has demonstrated that the identification of written words is influenced by their orthographic neighbors. Determining the orthographic neighbors of a word requires the ability to compute distances between any pairs of words. This is relatively straightforward to do in a language such as English, because most words do not have a hierarchical structure and characters are not compositional. It is far more difficult to do for Chinese characters.

Section 2 introduces related work on how diacritics and spatial relations influence word processing in various writing systems, providing theoretical background to support their explicit representations. Section 3 demonstrates a practical appli-

[1] Although the term *ideographic* is widespread, it is inaccurate. Chinese writing, specifically, is considered morphosyllabic (DeFrancis, 1989; Gorman and Sproat, 2023).

cation of IDSs in measuring distance between Chinese characters.

2. Related Work

2.1. Visual Processing of Diacritics

The use of diacritics is probably the best known application of compositionality in writing characters. A diacritic is usually defined as a glyph added to a character for pronunciation modification (Daniels and Bright, 1996). Evidence suggests that the processing of characters with diacritics depends on language features (Labusch et al., 2023). Ayçiçeği and Harris (2002) conducted a rapid serial visual presentation (RSVP) experiment in Turkish, showing more repetition blindness for words differing in a diacritic (işim- isim) as opposed to orthographic neighbours (ilim - isim), suggesting that characters with and without diacritics share the same mental representation. Perea et al. (2016) demonstrated that diacritic marks were quickly processed by the cognitive system during the early stages of word processing in Arabic, a script that is characterized by diacritical marks, position-dependent allography, and its cursive nature. Chetail and Boursain (2019), on the other hand, found that diacritic letters did not share the same abstract representations with their pure counterparts in French, where diacritic marks are predominantly observed on vowels. Marcet et al. (2022) found similar evidence for diverging abstract representations in Catalan, a language with complex grapheme-to-phoneme mappings.

2.2. Modeling Compositionality in Visual Processing

The Recognition by Components model (Biederman, 1987), asserted the significance of structural representations in object recognition. According to the model, the visual system recognizes an object by analyzing spatial arrangements of basic geometric shapes, such as cubes and cones. Transferring this to the domain of character recognition (Grainger et al., 2008), the relative positioning of components in a character is an important indicator of visual characteristics, helping to distinguish between characters (Lu et al., 2002). For example, the Chinese character 音 *yin1* 'sound' is distinguishable from another 昱 *yu4* 'bright' only by the relative positioning of components. The same is true for the diacritic letters ṡ and ṣ in the orthography of Yoruba in Nigeria. Arguably, even letters of Ъ and Б in Cyrillic script are compositionally similar, with a more nuanced difference in line orientation.

Other models focusing on the function of spatial relations in visual processing include Gestalt principles (Köhler, 1967; Todorovic, 2008) and feature integration theory (Treisman and Gelade, 1980).

3. Character Distance in Simplified Chinese Script

Ideographic description sequences were created to encode the spatial arrangement of components for CJK Unified Ideographs.[2] Unicode 15.1 (Unicode Consortium, 2023) defines eighteen ideographic description characters (IDCs), twelve of which are commonly used (Table 1).

An IDS consists of an IDC followed by its arguments, which can be either ideographs or another IDC. For instance, the IDS for the character 英 *ying1* 'blossom' is ⿱艹央, where ⿱ signifies top-down arrangement of the arguments 艹 and 央. Because the number of arguments to an IDC is always known, IDSs allow for nesting and concatenation. The ability to nest IDCs makes it possible to render complex spatial arrangements. For instance, the IDS for the character 嚻 *xiao1* 'a mythic beast' is ⿲⿱口口頁⿱口口.

When considering *distance* between the simplified Chinese characters 芍 *shao2* 'peony', 顶 *ding3* 'roof', and 英 *ying1* 'blossom', one approach would be to say that they are all different characters. Another approach could consist of noting that 芍 and 英 are both vertically arranged and have the element 艹 in common, whereas 顶 has no similarities to the other two characters, either in layout or components.

Existing methods for character similarity for Chinese characters can be divided in two main types: stroke-based and, more commonly, component-based. An abundance of literature defines the degree of character similarity based on shared component(s). In single-component comparison (Leck et al., 1995; Chen and Juola, 1982; Yeh and Li, 2002; Perfetti and Zhang, 1991), radicals (e.g., 蕉 *jiao1* 'banana' & 荐 *jian4* 'to recommend') or phonetic components (e.g., 煤 *mei2* 'coal' & 谋 *mou2* 'to plan') are used. Less often, smaller stroke patterns (e.g., 兑 *dui4* 'to exchange' & 分 *fen1* 'to divide'; Liu and Lin, 2008) and structural information (e.g., 啄 *zhuo2* 'to peck' & 偌 *ruo4* 'such'; Yeh et al., 1997) are used. Most of these methods define similarity in a binary way: a pair of characters is either similar or it is not. In the following sections, we propose an alternative method which is based on using edit distance on fully-decomposed IDSs and compare it to the existing approaches.

[2]CJK Unified Ideographs refers to a shared set of characters used in the writing systems of Chinese, Japanese, and Korean languages, all of which incorporate Han characters and their variations. CJKV extends the scope to include Vietnamese, which historically used Han characters.

IDC	Unicode	Name	Example	Example IDS
⿰	U+2FF0	Ideographic Description Character Left to Right	作	⿰亻乍
⿱	U+2FF1	Ideographic Description Character Above to Below	思	⿱田心
⿲	U+2FF2	Ideographic Description Character Left to Middle and Right	街	⿲彳圭亍
⿳	U+2FF3	Ideographic Description Character Above to Middle and Below	帚	⿳⺕冖巾
⿴	U+2FF4	Ideographic Description Character Full Surround	回	⿴囗口
⿵	U+2FF5	Ideographic Description Character Surround from Above	网	⿵冂⿻乂乂
⿶	U+2FF6	Ideographic Description Character Surround from Below	凶	⿶凵乂
⿷	U+2FF7	Ideographic Description Character Surround from Left	区	⿷匚乂
⿸	U+2FF8	Ideographic Description Character Surround from Upper Left	庆	⿸广大
⿹	U+2FF9	Ideographic Description Character Surround from Upper Right	句	⿹勹口
⿺	U+2FFA	Ideographic Description Character Surround from Lower Left	这	⿺辶文
⿻	U+2FFB	Ideographic Description Character Overlaid	巫	⿻工从

Table 1: The table provides IDCs, their Unicode, names, example characters, and Ideographic description sequences for the character.

3.1. Character distance using fully decomposed IDSs

We retrieved a dataset with IDSs for Chinese characters from an online repository[3], which in turn was derived from the Character Information Service Environment (CHISE) IDS database[4] (Morioka and Wittern, 2002). Part of the open-source CHISE project to expand general-purpose coded character sets, the IDS database contains most of the CJKV Unified Ideographs of ISO/IEC 10646 (Morioka, 2015).

To limit the list to Chinese characters used in mainland China, we selected only the 20,830 characters documented in Xinhua Dictionary (http://xh.5156edu.com). Then, we normalized the selected IDSs by recursively replacing components that could be further decomposed by their corresponding IDS. The result was a set of fully decomposed IDSs. Inspection of the resulting IDSs showed that, in addition to the 12 IDCs, only 545 basic characters were required to encode the over 20,000 selected characters.

Levenshtein distance (LD) is defined as the number of insertions, deletions, and substitutions operated on a string to turn it into another string (Levenshtein, 1966). Inspired by Kruskal (1983), we gave the substitution a cost of 2 and the other two operations a cost of 1 (see Figure 2).

3.1.1. IDS distance vs methods based on shared components

Some Chinese characters incorporate the same radicals and residuals: 案 *an4* 'instance' & 桉 *an1* 'the eucalyptus tree', 召 *zhao4* 'to summon' & 叨 *dao1* 'to chatter', and 峯 *feng1* 'peak' & 峰 *feng1* 'summit'. When similarity is based on shared components as in the examples (i.e., 吃 *chi1* 'to eat',

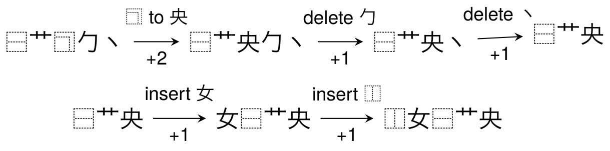

Figure 2: The first example illustrates substitution and deletion. Converting 芍 *shao2* 'peony' (IDS: ⿱艹⿹勹丶) to 英 *ying1* 'blossom' (IDS: ⿱艹央) involves one substitution and two deletions, resulting in an edit distance of 4. The second example illustrates insertion: Transforming 英 to 媖 *ying1* beauty (IDS: ⿰女⿱艹央) requires the insertion of ⿰ and 女, resulting in an edit distance of 2.

员 *yuan2* 'member', 哲 *zhe2* 'philosophical', and 加 *jia1* 'to add') provided in the work of Yeh and Li (2002), these pairs are identical because they all have the same components. This method falls short with respect to structural differences.

Figure 3 shows the IDS distance for the same characters. The IDS distance between 案 and 桉 is 4, equal to that between 召 & 叨, whereas closer are 峯 & 峰, which are only different in layout and have a distance of 2.

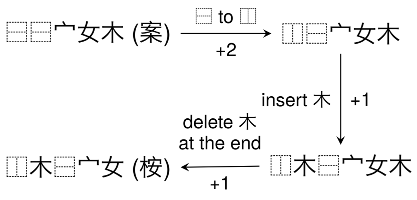

Figure 3: The edit distance of 案 & 桉 is 4, summing up 1 substitution, 1 insertion and 1 deletion.

[3] https://github.com/cjkvi/cjkvi-ids
[4] https://gitlab.chise.org/CHISE/ids

3.1.2. IDS distance vs methods based on radical-level shared components and character structures

The spatial arrangements of components in Chinese characters are highly correlated with their functions (semantic or phonetic). In various applications, characters that have identical structure and shared components are considered similar and selected as stimuli (Leck et al., 1995; Chen and Juola, 1982). Hence, these methods do not identify similarity among characters sharing both structure and components, but not the function of components. For instance, 杏 *xing4* 'apricot' and 呆 *dai1* 'dull' have different component order and semantic radical, but are otherwise identical. Figure 4 shows how, in comparison, IDS-based distance addresses this.

$$\text{⿱木口 (杏)} \xrightarrow[+1]{\text{delete 木}} \text{⿱口} \xrightarrow[+1]{\substack{\text{insert 木}\\ \text{at the end}}} \text{⿱口木 (呆)}$$

Figure 4: The edit distance of 杏 & 呆 is 2 via one deletion and one insertion.

3.1.3. IDS distance vs methods based on sub-radical shared components and character structures

Liu and Lin (2008) go beyond the radical - residual level and explore smaller stroke patterns in computing similarity between Chinese characters. They decompose a character into a set of 24 basic elements defined in the Cangjie code by Chu (1979). A character is represented by its structure (one of nine layout patterns, encoded as a real value) followed by up to three components. For example, 相 *xiang4* 'appearance' has a representation of '2 (layout code) - 木 (part 1) - 月山 (part 2)'. Although this approach goes a long way toward addressing the compositionality of characters in a principled manner, the limited basic components do not allow for an unambiguous specification of characters. In other words, in many cases the decomposition does not allow for recomposition of the original characters. Using only nine layout patterns is also insufficient, as Simplified Chinese characters can be as complex as encompassing up to 32 strokes (e.g, 龘 *da2* 'depicting the majestic soaring of a dragon'). Instead, using IDS, we encode character structures by explaining structural information between just two or three components predetermined by IDCs. This granularity is also a reason why our method produce faithful results.

The IDS representation does not have structural ambiguity between sequences. In the few cases where we have found characters to share the same IDS representation (56 out 20,830), this

$$\text{⿰木目 (相)} \xrightarrow[+1]{\text{insert 竹}} \text{竹⿰木目} \xrightarrow[+1]{\text{insert ⿱}} \text{⿱竹⿰木目 (箱)}$$

$$\text{⿰木目 (相)} \xrightarrow[+1]{\text{insert ⿱}} \text{⿱⿰木目} \xrightarrow[+1]{\text{insert 心}} \text{⿱⿰木目心 (想)}$$

$$\text{⿱竹⿰木目 (箱)} \xrightarrow[+1]{\text{delete 竹}} \text{⿱⿰木目} \xrightarrow[+1]{\text{insert 心}} \text{⿱⿰木目心 (想)}$$

Figure 5: The figure shows the process to get edit distance among characters 相, 箱, and 想. We turn 相 into 箱 via two insertions, as is the case for 相 & 想. Converting 箱 to 想 requires one deletion and one insertion.

concerned historical variants of the same character with slightly different stroke variants. Figure 5 shows the IDS distance between 相 *xiang4* 'appearance', 箱 *xiang1* 'case', and 想 *xiang3* 'to miss', which according to Liu and Lin (2008) would be dissimilar. It may seem to be surprising that the IDS distance between 箱 and 想 is so small, but, in addition to overlapping components, these two are both vertical characters hierarchically enclosing a horizontal character.

3.1.4. Three basic elements to compute character similarity

It seems that, from the above-mentioned examples, it is next to impossible to evaluate the degree of character similarity without the integration of three basic elements:

- components (stroke patterns or smaller), as opposed to single-component comparison.
- layouts governing components, in comparison with character structure possibly as a result of single component comparison.
- relative positions of components.

We suggest that if we want to claim that characters are similar, we need to make these three elements explicit.

One of the limitations of using IDS to compute edit distance is its comprised ability to differentiate relative positions of components in some cases. For example, characters 呆 *dai1* 'dull', 宋 *song4* 'a surname', and 告 *gao4* 'to sue' have the same edit distance of 2, but while we rule in component position, 告 should be further away from 呆, as their shared component is located differently. In order to differentiate the effect of component order, one possible method is to increase the weight of insertion. However, this would lead to an asymmetry of pairwise distances, which requires further modification.

杳		鬱	
Neighbors	Distance	Neighbors	Distance
杳	0	鬱	0
查	2	鬰	6
査	2	爩	10
杏	2	鬯	12
杚	2	楙	18
杢	2	僂	19
李	2	欆	20
旮	2	鑀	20
木	2	蓾	20
杰	2	滷	20
桼	2	燓	20
昷	2	樐	20
呆	2	樊	20
杲	2	鐢	20
晏	2	鹵	20
柰	2	乘	21
早	2	兇	21
旯	2	亶	21
日	2	爻	21
旦	2	壱	21

Table 2: Neighbors and their pairwise distance to 杳 (orthographically simple and dense) and to 鬱 (orthographically complex and distinct).

3.2. Character Neighbors

By computing character distance, it is possible to cluster orthographically similar characters by exhausting pairwise distances among all characters and sorting the result. Table 2 provides twenty nearest neighbors for 杳 *yao3* 'dim' and 鬱 *yu4* 'lush and growing abundantly'.

However, this could be problematic, as distance is modulated by sequence length. For example, character 鬱 is closer to 匕 *bi3* 'spoon' (distance: 23) than 礬 *fan2* 'alum' (distance: 24), although the latter may seem to be more similar due to identical structures and more common components. The reason is that 匕 (IDS: ⿺乚丿, length: 3) requires only addition to transform into the target character 鬱 (IDS: ⿳⿲木⿱乂⿻丿乀木冖⿰⿱⿵凵⿻乂⿳丶⿰丶丶丶⿺乚丿彡, length: 26), while 礬 (IDS: ⿱⿱⿲木⿱乂⿻丿乀木⿻一人⿸⿱一丿口, length: 18), with longer sequences, requires additional deletion.

To address this, we normalize distance as follows: Let N_i be the length of IDS of character C_i and N_j be the length of IDS of character C_j:

$$max\ distance = \min(N_i, N_j) \times 2 + |N_i - N_j|$$

where *max distance* represents the upper bound of the cost from possible operations. The distance metric can then be normalized by calculating the relationship between the cost of operations actually used and the maximum possible costs. The normalized distance is calculated as:

$$normalized\ distance = \frac{edit\ distance}{max\ distance}$$

where *normalized distance* indicates a measure where lower values signify higher operational congruence and thus closer distance.

Twenty closest neighbors for 鬱 based on normalized distance are given in Table 3. Note that the normalized distance of 鬱 and and 礬 is 0.545, smaller than that of 鬱 and 匕, though not shown in the table, at 0.793. We can see that the adjusted distance reveals a cluster of character pattern that is closer to human intuition.

4. IDS as a general approach to expressing component relations in any script

There are some advantages of the ideographic description sequences. First, they have the potential to be used to describe attested compositional characters in any script. For instance, French *café* could be represented as (c, a, f, ⿱, ´, e), with the description character ⿱ indicating that the two subsequent elements are to be arranged top-down. Second, they provide the possibility to form new compositional characters. Finally, when words are represented using concatenated ideographic description sequences, they allow for more accurate measurement of word similarity. For instance, in French, the word pâte can be considered to differ by one character from both pate and pâté, but in the same way it can also be considered to differ by the absence or presence of a diacritic on one of the characters.

In practice, the arguments to a particular IDC are quite predictable. For example, the IDC ⿰ almost always has a semantic radical as its left component and a phonetic residual as its right component. The spatial rendering thus typically also encodes a specific relationship. Taking this idea further, we can consider an IDC as a way of connecting a specific type of linguistic relationship to a specific spatial rendering (e.g., morphological, semantic, ontological). For instance, Table 4 shows how one could consider the components of compound words as arguments to a relationship operator which horizontally concatenates the components. This would allow distinguishing compounds from non-compounds, at least in the underlying sequence. But instead of horizontal arrangement, we could also replace the horizontal IDC with an equivalent vertical IDC to achieve a different kind of representation. In some applications, English text could then be rendered as in Figure 6. On top of this, there are also a wide range of other possible applications for IDS: creating novel

Neighbors	鬱	鬰	爩	[illegible]	鏤	僂	樓	蓾	滷	樐
Normalized Distance	0	0.125	0.192	0.3	0.442	0.463	0.463	0.5	0.5	0.5
Neighbors	磠	塷	鑥	燓	樊	鐢	鹵	棥	[illegible]	鹷
Normalized Distance	0.5	0.524	0.524	0.526	0.526	0.526	0.526	0.529	0.545	0.545

Table 3: Twenty neighbors to 鬱 based on normalized distance. Note that after adjusting for the complexity level of the characters, the result is closer to human intuition.

Compound word	**IDS horizontal**	**IDS vertical**	**Representation vertical**
red dwarf	⿲, red, , dwarf	⿱, red, dwarf	red dwarf
red-blooded	⿲, red, -, blooded	⿱, red, blooded	red blooded
redhead	⿰, red, head	⿱, red, head	red head

Table 4: Table shows three compound words, IDS for their original forms, IDS for vertical placements, and resulting vertical renditions.

- Red dwarfs are the most common type of star in the Milky Way.
- He says he's a red blooded American male!
- Unusually for a red head , she tans easily.

Figure 6: A demonstration of rendering compound words in vertical layouts. Example sentences were retrieved from online Cambridge Dictionary (https://dictionary.cambridge.org).

sequences; creating or adapting representations for under-resourced languages; rendering linguistic relationships spatially; substituting layouts, etc.

Examples of character IDS application in different scripts are shown in Table 5. While we use existing IDCs designed for Simplified Chinese characters in these examples, specific IDCs may need to be created to allow for script characteristics.

5. Conclusion

In this paper, we demonstrated that IDSs can be used to more precisely calculate edit distance and orthographic neighbors for Simplified Chinese characters. In addition, we explored the possibility of using IDSs to typographically represent morphological relationships. While Unicode currently only uses IDSs for CJK writing systems, the ability to represent characters compositionally gives IDSs a wide range of application beyond these scripts. In this way, representing characters and words using IDSs can offer methodological improvements in several areas.

Character	Script / Language	IDS
ɯ̌	Adlam	⿱, ◌̌, ɯ
Ъ̀	Cyrillic	⿱, ` , Ъ
님	Korean	⿱, ⿰, ㄴ, ㅣ, ㅁ
Ͽ	Greek	⿴ , ɔ, ·
Ŀ	Latin	⿺, L , ·
Đ	Latin	⿻, D, -
Ⱡ	Latin	⿻, L, ⿱, -, -
Æ	Latin / Cyrillic	⿰, A, E
Ǣ	Latin	⿱, -, ⿰, A, E
Ǟ	Latin	⿳, -, ◌̈, A
ஊ	Tamil	⿺, உ, ள

Table 5: Example characters represented as IDS in several scripts like Adlam, Cyrillic, Korean, Greek, Latin, and Tamil.

References

American National Standards Institute. 1995. 7-bit American national standard code for information interchange. *Standards Action*. Retrieved February 6, 2024, from https://webstore.ansi.org/standards.

Ayse Ayçiçeği and Catherine L. Harris. 2002. How are letters containing diacritics represented? Repetition blindness for Turkish words. *European Journal of Cognitive Psychology*, 14(3):371–382.

Irving Biederman. 1987. Recognition-by-components: a theory of human image understanding. *Psychological review*, 94(2):115.

Hsuan-Chih Chen and James F Juola. 1982. Dimensions of lexical coding in Chinese and English. *Memory & Cognition*, 10:216–224.

Fabienne Chetail and Emeline Boursain. 2019. Shared or separated representations for letters with diacritics? *Psychonomic Bulletin & Review*, 26(1):347–352.

Bong-Foo Chu. 1979. Laboratory of Chu Bong-Foo. Retrieved Februrary 6, 2024, from http://www.cbflabs.com.

Peter T Daniels and William Bright. 1996. *The world's writing systems*. Oxford University Press.

John DeFrancis. 1989. *Visible speech: The diverse oneness of writing systems*. University of Hawaii Press.

Kyle Gorman and Richard Sproat. 2023. Myths about writing systems in speech & language technology. In *Proceedings of the Workshop on Computation and Written Language (CAWL 2023)*, pages 1–5.

Jonathan Grainger, Arnaud Rey, and Stéphane Dufau. 2008. Letter perception: from pixels to pandemonium. *Trends in Cognitive Sciences*, 12(10):381–387.

ISO/IEC. 1987. ISO/IEC 8859: 8-bit character encodings. International Organization for Standardization and International Electrotechnical Commission.

Wolfgang Köhler. 1967. Gestalt psychology. *Psychologische Forschung*, 31(1):XVIII–XXX.

Joseph B Kruskal. 1983. An overview of sequence comparison: Time warps, string edits, and macromolecules. *SIAM review*, 25(2):201–237.

Melanie Labusch, Stéphanie Massol, Ana Marcet, and Manuel Perea. 2023. Are goats chèvres, chévres, chēvres, and chevres? Unveiling the orthographic code of diacritical vowels. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 49(2):301–319.

Kwong Joo Leck, Brendan S Weekes, and May Jane Chen. 1995. Visual and phonological pathways to the lexicon: Evidence from Chinese readers. *Memory & Cognition*, 23:468–476.

Vladimir Iosifovich Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710.

Yi-Chen Lin, Hsiang-Yu Chen, Yvonne C Lai, and Denise H Wu. 2015. Phonological similarity and orthographic similarity affect probed serial recall of Chinese characters. *Memory & Cognition*, 43:538–554.

Chao-Lin Liu and Jen-Hsiang Lin. 2008. Using structural information for identifying similar Chinese characters. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies Short Papers - HLT '08*, page 93, Columbus, Ohio. Association for Computational Linguistics.

Qin Lu, Shiu Tong Chan, Yin Li, and Ngai Ling Li. 2002. Decomposition for ISO/IEC 10646 ideographic characters. In *COLING-02: The 3rd Workshop on Asian Language Resources and International Standardization*.

Ana Marcet, María Fernández-López, Ana Baciero, Albert Sesé, and Manuel Perea. 2022. What are the letters e and é in a language with vowel reduction? The case of Catalan. *Applied Psycholinguistics*, 43(1):193–210.

Tomohiko Morioka. 2015. Multiple-policy character annotation based on chise. *Journal of the Japanese Association for Digital Humanities*, 1(1):86–106.

Tomohiko Morioka and Christian Wittern. 2002. Developping of character object technology with character databases. *IPA result report*.

Manuel Perea, Reem Abu Mallouh, Ahmed Mohammed, Batoul Khalifa, and Manuel Carreiras. 2016. Do Diacritical Marks Play a Role at the Early Stages of Word Recognition in Arabic? *Frontiers in Psychology*, 7.

Charles A Perfetti and Sulan Zhang. 1991. Phonological processes in reading Chinese characters. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 17(4):633.

I-Fan Su, Sin-Ching Cassie Mak, Lai-Ying Milly Cheung, and Sam-Po Law. 2012. Taking a radical position: evidence for position-specific radical representations in Chinese character recognition using masked priming erp. *Frontiers in Psychology*, 3:333.

Dejan Todorovic. 2008. Gestalt principles. *Scholarpedia*, 3(12):5345.

Anne M Treisman and Garry Gelade. 1980. A feature-integration theory of attention. *Cognitive psychology*, 12(1):97–136.

Unicode Consortium. 2023. *The Unicode Standard, Version 15.1.0*. The Unicode Consortium, South San Francisco, CA. ISBN 978-1-936213-33-7.

Senlin Xu, Mingfan Zheng, and Xinran Li. 2020. String comparators for Chinese-characters-based record linkages. *IEEE Access*, 9:3735–3743.

Su-Ling Yeh and Jing-Ling Li. 2002. Role of structure and component in judgments of visual similarity of Chinese characters. *Journal of Experimental Psychology: Human Perception and Performance*, 28(4):933.

Su-Ling Yeh, Jing Ling Li, I Chen, et al. 1997. The perceptual dimensions underlying the classification of the shapes of Chinese characters. *Chinese Journal of Psychology*.

Pong Chi Yuen, Guo-Can Feng, and Yuan Yan Tang. 1998. Printed Chinese character similarity measurement using ring projection and distance transform. *International journal of pattern recognition and artificial intelligence*, 12(02):209–221.

Xiaochen Zhang, Siqin Yang, and Minghu Jiang. 2020. Rapid implicit extraction of abstract orthographic patterns of Chinese characters during reading. *Plos one*, 15(2):e0229590.

Author Index