CAMELMORPH MSA: A Large-Scale Open-Source Morphological Analyzer for Modern Standard Arabic

Christian Khairallah,[†] Salam Khalifa,[†,††] Reham Marzouk,[†,‡] Mayar Nassar,[†,‡‡] Nizar Habash[†]
Computational Approaches to Modeling Language (CAMeL) Lab
[†]New York University Abu Dhabi, [††]Stony Brook University
[‡]Alexandria University, [‡‡]Ain Shams University
{christian.khairallah,nizar.habash}@nyu.edu, salam.khalifa@stonybrook.edu,
igsr.r.marzouk@alexu.edu.eg, mayar.nassar@art.asu.edu.eg

Abstract

We present CAMELMORPH MSA, the largest open-source Modern Standard Arabic morphological analyzer and generator. CAMELMORPH MSA has over 100K lemmas, and includes rarely modeled morphological features of Modern Standard Arabic with Classical Arabic origins. CAMELMORPH MSA can produce ∼1.45B analyses and ∼535M unique diacritizations, almost an order of magnitude larger than SAMA (Maamouri et al., 2010c), in addition to having ∼36% less OOV rate than SAMA on a 10B word corpus. Furthermore, CAMELMORPH MSA fills the gaps of many lemma paradigms by modeling linguistic phenomena consistently. CAMELMORPH MSA seamlessly integrates with the Camel Tools Python toolkit (Obeid et al., 2020), ensuring ease of use and accessibility.

Keywords: Arabic, Morphology, Open-Source

1. Introduction & Motivation

Arabic presents many challenges to natural language processing (NLP), ranging from its hybrid templatic and concatenative morphology, rich collections of inflectional features, to numerous allomorphs, highly ambiguous orthography, and dialectal variants. Over the last four decades, many approaches have been explored in developing Arabic morphological analyzers and generators (Beesley et al., 1989; Kiraz, 1994; Buckwalter, 2004; Smrž, 2007b; Maamouri et al., 2010c; Boudchiche et al., 2017; Taji et al., 2018; Habash et al., 2022). These tools continue to show value for Arabic NLP when paired with state-of-the-art neural models on tasks such as morphological tagging (Zalmout and Habash, 2017; Inoue et al., 2022), sentiment analysis (Baly et al., 2017), controlled text rewriting (Alhafni et al., 2022), and grammatical error correction (Alhafni et al., 2023). Developing such tools is neither cheap nor easy, and many of them are either not open-source, not freely available, or incomplete.

In this paper, we present **CAMELMORPH MSA**, a part of the CAMELMORPH Project,[1] which is a large effort to develop morphological analyzers and generators under a common framework for a number of Arabic variants: Modern Standard Arabic (MSA) فصحى العصر, Classical Arabic (CA) فصحى التراث and Arabic Dialects (DA) اللهجات العربية. CAMELMORPH MSA builds on learned lessons and available tools to create the largest to date open-source Arabic morphological analyzer and generator in terms of morphological coverage. As part of developing CAMELMORPH MSA, we extend the CAMELMORPH framework introduced by Habash et al. (2022), making improvements to its design and implementation. We also expand upon the lexical and morphological resources reported in Habash et al. (2022) and Khairallah et al. (2024), to include missing parts-of-speech (POS) and rarely modeled MSA and CA morphological features. The result is a very large analyzer with over 100K lemmas, producing ∼1.45B analyses for ∼535M diacritizations, almost an order of magnitude larger than SAMA (Maamouri et al., 2010c). We make all our resources, code, guidelines, and documentation publicly available.[1]

2. Related Work

This work builds on a long history of morphological analysis and generation tools (Al-Sughaiyer and Al-Kharashi, 2004; Habash, 2010; Alothman and Alsalman, 2020; Tachicart et al., 2022). Altantawy et al. (2011) categorize these systems on a spectrum reflecting their morphological representation modeling approaches. At one end, the representations are characterized by abstraction and a greater reliance on a templatic-affixational perspective of morphology (Beesley et al., 1989; Kiraz, 1994; Beesley, 1996; Habash and Rambow, 2006; Smrž, 2007a; Boudchiche et al., 2017), while at the other end, they adopt a more derivation-inflectional driven and surfacy approach (Buckwalter, 2004; Maamouri et al., 2010c; Taji et al., 2018). The former tends to rely on multi-tiered representa-

[1] http://morph.camel-lab.com.

LREC-COLING 2024, pages 2683–2691
20-25 May, 2024.

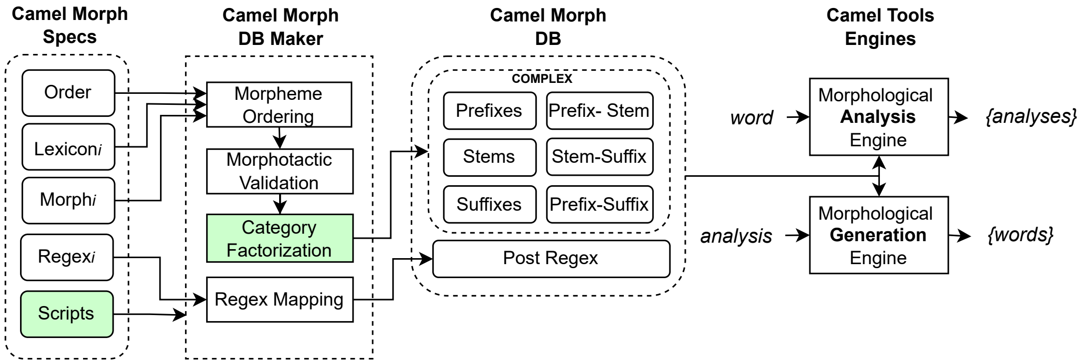

Figure 1: A high-level diagram of the CAMELMORPH framework used to develop CAMELMORPH MSA.

tions that map lexical to surface forms, generally in the form of a finite-state transducer through complex rules and can either model at the morpheme (Beesley, 1996) or lexeme level (Smrž, 2007a). The latter tends to follow a more lexicon and stem-based approach where morphotactic rules are built directly into the lexicon and inherently models at the morpheme and features level, without including roots and patterns into the rules. The most widely used of these models rely on the Buckwalter six-table approach (Buckwalter, 2004), which entails a lexicon of morphemes and compatibility tables.

This paper builds on the CAMELMORPH framework (Habash et al., 2022), which is aligned with the stem-based methodologies, but leverages morphotactic allomorphy modeling via inter-allomorphic compatibility rules and uses a lexicon that is comparatively easy to modify and extend. We consult and expand on many resources (Buckwalter, 2004; Maamouri et al., 2010c; Taji et al., 2018; Habash et al., 2022; Khairallah et al., 2024).

3. The CAMELMORPH Framework

Overview CAMELMORPH MSA is developed within the CAMELMORPH framework, which relies on a two-step process to perform morphological analysis and generation (Figure 1) (Habash et al., 2022). The first step requires designing morphological specifications (**CAMELMORPH Specs**) describing the language's grammar and lexicon in a manner which is annotator friendly, which are then converted via an offline process powered by its **DB Maker** algorithm into a morphological database (**CAMELMORPH DB**) in the six-table style of the Buckwalter Arabic Morphological Analyzer (BAMA) DBs (Buckwalter, 2004; Maamouri et al., 2010c; Taji et al., 2018). The created DBs can be used by any compatible analysis and generation engine, e.g., Camel Tools (Obeid et al., 2020).

The morphological specifications can be divided into (a) open-class **Lexicon** (lemmas and stems), (b) closed-class **Morph** (affixes and clitics) and (c) **Order** specifications. **Lexicon** and **Morph** consist of *allomorphs* organized into *morphemes*. And **Order** sequences specify the positions of all *morpheme classes* in a word. The lexicon is a large repository that contains stems, their associated lemmas, and other features. Associated with each allomorph is a set of hand-crafted *conditions*, which control the allomorph selection of specific morphemes. The offline **DB Maker** process makes heavy use of these conditions to determine proper combinations and compatibility among the allomorphs in a word. Finally, the framework accommodates the use of ortho-phonological rewrite regex rules (such as sun-letter handling or epenthesis) as part of the specifications to be used by the analysis and generation engines (**Post Regex**). See Habash et al. (2022) for more details.

Framework Extensions As part of the effort presented here, we introduced two extensions to CAMELMORPH (highlighted in Figure 1).

First is ***Category Factorization***. Each complex morpheme (Prefix, Stem, and Suffix) in the database is given a *category* which controls what other complex morphemes it can pair up with during analysis and generation. These categories are generated by the **DB Maker** using condition and morpheme class information, and are very strongly-typed, and thus, very redundant. Our extension maps all compiled categories with the same dependencies to each other. This improves the speed of analysis by reducing the number of categories and compatibility combinations.

Second is ***Specification Scripts***. We extend the original CAMELMORPH framework to accept scripts that preprocess specifications before feeding them to the **DB Maker**. This feature is useful when specifications can be created automatically using an algorithm which does not require human intervention (see more details in §4).

Next, we present the CAMELMORPH MSA DB we created using this framework.

		Count	Verb Example	Count	Noun Example	Count	Particle Example
Morphology Specifications	Lemma	1	**ramaý** 'throw' **رَمَى**	1	**safiyr** 'embassador' **سَفِير**	1	**çalaý** 'on, upon' عَلَى
	Proclitics	30	wa, fa, **Âa**, ... وَ، فَ، **أَ**، ...	19	Al, **wa**, fa, **li**, bi, ... ال، **وَ**، فَ، **لِ**، بِ، ...	9	wa, **fa**, Âa, ... وَ، **فَ**، أَ، ...
	Prefixes	13	ya, ta,... يَ، تَ، ...	N/A		N/A	
	Pre-Buffers	1	a ◌َ	N/A		N/A	
	Stems	3	**رَم (ماضي)**، رْم (مضارع)، رْم (أمر) **ram (perfect)**, r.m (imperfect/command)	2	**سَفِير**، سُفَرَا **safiyr** (base), sufaraA (broken plural)	4	عَلَى، **عَلَيْ**، عَلَامَ،... çalaý, **çalay.**, çalaAma,...
	Post-Buffers	11	aý, ∅, **ay**, iy, ... ◌َى، ∅، **◌َي**، ◌ِي، ...	5	ŵ, ', ŷ, ∅, ... ؤ، ء، ئ، ∅، ...	0	
	Suffixes	100	tu, **ta**, naA, at, ... تُ، **تَ**، نَا، ◌َت، ...	79	a, i, ũ, aħu, **aAti**, ... ◌َ، ◌ِ، ◌ٌ، **◌َاتِ**،◌َةُ،...	0	
	Enclitics	18	**hu**, hi, hum, him ... **هُ**، هِ، هُم، هِم، ...	19	hu, hi, hum, **him** ... هُ، هِ، هُم، **هِم**، ...	13	hu, hi, hum, **haA**, ... هُ، هِ، هُم، **هَا**، ...
	Conditions	*25*	**#ay** (defective),...	25	**FP** (+At suffix),...	6	**enc0** (presence of object clitic)
	Order Seqs.	69	**verbal orders**	21	**nominal orders**	2	**particle orders**
DB	Complex Prefixes	3,129	wa, fa, **Âa**, waya, ... وَ، فَ، **أَ**، وَيَ، ...	199	wa, waAl, **wali**, lil, ... وَ، وَال، **وَلِ**، لِل، ...	14	wa, **fa**, >a, >awa, ... وَ، **فَ**، أَ، أَوَ، ...
	Complex Stems	10	ramaý, **ramay.**, ... رَمَى، **رَمَيْ**، ...	4	**safiyr**, sufarAŵ, ... **سَفِير**، سُفَراؤ، ...	4	çalaý, **çalay.**, çalaAma,... عَلَى، **عَلَيْ**، عَلَامَ،...
	Complex Suffixes	972	ta, **tahu**, hi, ... تَ، **تَهُ**، هِ،...	243	aAtu, **aAtihim**, aki,... ◌َاتُ، **◌َاتِهِم**،◌َكِ، ...	14	hu, hi, hum, **haA** ... هُ، هِ، هُم، **هَا**، ...
	Compatibility Combinations	2,687	**Stem-Suffix**, **Prefix-Stem**, **Prefix Suffix**	394	**Stem-Suffix**, **Prefix-Stem**, **Prefix Suffix**	14	**Stem-Suffix**, **Prefix-Stem**, **Prefix Suffix**
Word Forms (unique analyses)		42,588	Word: **Âaramay.tahu أَرَمَيْتَهُ** `did you throw it?' Features: **verb**, **perfective**, **active**, **2nd person**, **masculine**, **singular**, **interrogative**, **3rd person ms direct object**	12,942	Word: **walisafiyraAtihim وَلِسَفِيرَاتِهِم** `and for their ambassadors' Features: **noun**, **feminine**, **plural**, **genitive**, **construct**, **3rd person mp possessive**, **conjunction**, **preposition**	224	Word: **façalay.hA فَعَلَيْها** `so, upon it' Features: **prep**, **3rd person fs pron**, **conjunction**

Table 1: Examples of the various specifications needed for morphological modeling. The colored items participate in forming the full word in the last row. HSB Arabic transliteration (Habash et al., 2007).

4. CAMELMORPH MSA

Logistics The CAMELMORPH Project team comprises five computational linguists, all Arabic native speakers, who participated in the design and annotation of CAMELMORPH MSA. This multi-year effort was reported on incrementally in Habash et al. (2022) and Khairallah et al. (2024). We make in-depth technical guidelines publicly available.[1]

Morphological Specifications We classify the elements specified in CAMELMORPH MSA Specs into: ***essential*** and ***supplementary***.

Essential elements are core components for morphological modeling in the CAMELMORPH approach. They include: (a) **Order** specifications, (b) **morpheme class** which specifies the order sequence a morpheme can be paired with, (c) **lemma**, (d) **stem and morph form**, (e) functional/form-based **morphological features**, and (f) *required* and *set* **conditions**.

Supplementary elements are additional dependent information which we added to match the expected performance of other systems, and in some cases we make use of them to automatically extend the essential elements. They include: (a) **root**, **abstract pattern**, and **concrete pattern** (concrete patterns are used for interdigitation purposes, as opposed to the abstract patterns which abstract away over groups of concrete patterns), (b) **transcription**, provided in CAPHI format (Habash et al., 2018), (c) **tokenization and segmentation**, provided in the D3 and ATB schemes (Habash, 2010), (d) **English gloss**, and finally, (e) **lemma and lemma+POS likelihood** (log-probability). See Appendix A.

Development Process As part of the development of CAMELMORPH MSA, we employed a variety of techniques, spanning from manual to automated approaches, to extract, build, correct, and quality check the CAMELMORPH MSA Specs.

We manually specified all affixes and clitics, and their sequence **orders**, as well as the morphological **conditions** needed to model MSA, with an eye towards future modeling of dialectal Arabic.

The work on the **Lexicon** started with automatically extracting lemmas, their stems, and their features from publicly available resources, such as $\text{CALIMA}_{\textit{Star}}$ (Taji et al., 2018), which extends on SAMA (Maamouri et al., 2010) (henceforth, SAMA/CALIMA). We also extracted a large collection of names of people, places, and organizations from Wikidata. In an extensive process, we manually modified all lexical entries to fit within our morphological conditions and targeted full paradigms. Furthermore, all the English glosses, roots, and patterns were manually checked and their gaps remedied.

We use specification scripts to automate the addition of three elements. First, we automatically added CAPHI transcriptions into the compiled DB by conversion from stem and morph forms. Second, we automatically extended the verb lexicon by adding passive voice verb stems which are systematically derivable from their active voice counterparts. Finally, to extract likelihood information, we use the *training* portion of the Penn Arabic Treebank (*PATB-Train*, see §5 for details). We synchronized the data with our DB to identify the closest best match as there is a considerable mismatch in the spelling of our lemmas and the PATB's.

		Camel Morph MSA Specs	Camel Morph MSA DB	SAMA/CALIMA DB		
(a)	**Lemmas (Stems)**	105,102 (140,612)	105,102 (154,573)	42,218 (71,466)	**Lemmas (Stems)**	(c)
	Verbs	9,333 (38,156)	9,333 (47,540)	9,279 (26,343)	*Verbs*	
	Nominals	33,267 (39,837)	33,267 (44,414)	32,701 (44,742)	*Nominals*	
	Others	230 (347)	230 (347)	238 (381)	*Others*	
	Proper Nouns - Annex	62,272 (62,272)	62,272 (62,272)			
(b)	**Prefix Morphs (Allom.)**	60 (65)	14,726	4,640	**DBPrefix Sequences**	(d)
	Suffix Morphs (Allom.)	205 (406)	12,724	1,191	**DBSuffix Sequences**	
	Stem Buffers	111	15,044	979	**Compatibility Entries**	
	Unique Condition Terms	88	535,186,314 (242,824,398)	70,250,488	**Unique Diacritized Forms (no Annex)**	(e)
	Morph Order Sequences	122	1,447,312,125 (630,731,386)	227,471,211	**Unique Analyses (no Annex)**	
			8,070,764 (3,214,695)	1,514,577	**Unique Analyses w/o Clitics (no Annex)**	

Table 2: Statistics comparing CAMELMORPH MSA Specs and DB with the SAMA/CALIMA DB.

Quality Checking Given the size of this project and its many moving parts, we regularly quality checked our specifications using a number of techniques that isolate specific phenomena while freezing some elements such as the lemmas or POS. Specifically, we worked on debugging the **morphotactics** of the affixes and clitics, **lemma paradigm** completeness, and **stem subparadigm** correctness. We regularly made use of the Camel Tools generation engine (Obeid et al., 2020) to validate the analysis and generation processes; and we used different clustering techniques to group related phenomena into size-manageable sets for our annotators to debug. The process usually involved the assigned annotator marking wrong outputs and discussing them with the rest of the team to make the needed changes. All markings of wrong and correct outputs are *banked* so they can be used in later automatic progress evaluation.

Examples In Table 1, we show three lemma examples that summarize most of the framework's components. In the **Morphology Specifications** rows, we see the morpheme classes in order (as per our **Order** sequences). The counts pertain to the number of specifications (per specified unit) that participate in the number of unique analyses that can be generated for that specific lemma (last row in Table 1). After the specifications are converted by the **DB Maker** to DB entries, note how a relatively small amount of allomorphs is mapped into a large number of complex morphemes (in the **DB** Section). In particular, the *buffers*, which are small segments meant to complete other morphemes under different conditions (Habash et al., 2022), together with the passive voice *scripts* mentioned earlier, reduce the total number of stems that annotators have to manage at the specifications level: 10 generated from 3 specified for the verb and 4 generated from 2 specified for the noun. Finally, constrained by the compatibility combinations, the complex morphemes can combine into a very large number of analyses.

Statistics Table 2 compares the statistics of our specifications (CAMELMORPH MSA Specs) and their associated DB (CAMELMORPH MSA DB), with those of the SAMA/CALIMA DB (Maamouri et al., 2010; Taji et al., 2018). We first note the 10% increase in the number of stems between CAMELMORPH MSA Specs and CAMELMORPH MSA DB. This shows that the CAMELMORPH MSA Specs are able to compactly represent stems, denoting a more annotator-friendly morphological modeling, and is attributed to the buffer system which shifts the modeling weight to **Morph** from **Lexicon** (see §3). This can also be seen at the prefix/suffix, condition, and order levels where a small number of specifications (CAMELMORPH MSA Specs) leads to a large number of sequences (CAMELMORPH MSA DB) as seen in Table 2 (b) and (d).

We also note that the number of lemmas/stems for CAMELMORPH MSA DB is about 2.3 times greater than for SAMA/CALIMA DB. This is due to two main reasons: (a) the gaps of verbal paradigms we filled by adding command and passive voice stems, most of which were absent in SAMA/CALIMA; and (b) the 62K (mostly undiacritized) Wikidata entries, which we identify as *Proper Nouns - Annex* in Table 2.

Finally, the number of analyses that CAMELMORPH MSA DB can generate is almost 9 times greater than for SAMA/CALIMA DB, and this is reflected by the greater number of compatibility combinations, and complex prefixes/suffixes/stems. The number of unique analyses when clitics are not considered is about 8.1M for CAMELMORPH MSA DB (or 3.2M without *Proper Nouns - Annex*) versus 1.5M for SAMA/CALIMA DB. This shows that despite the high inflation that clitics may cause, our generative power is still superior (with and without *Proper Noun - Annex*). This large increase when including clitics is due to the modeling of various less frequent CA morphemes in MSA that are either not modeled completely or simply not modeled at all in SAMA/CALIMA DB, e.g., the interrogative proclitic, the energetic moods for verbs, and the indirect object pronominal clitics.

5. Experimental Results

In this section, we conduct experiments to evaluate the **speed**, **coverage**, and **accuracy** of CAMELMORPH MSA against CALIMA$_{Star}$ (Taji et al., 2018), an extended version of SAMA (Maamouri et al., 2010c) (henceforth, SAMA/CALIMA). Both DBs are accessed through the Camel Tools analysis/generation engines (Obeid et al., 2020).

Datasets We report using three datasets. First is *MSA-CB*, a set of 11.4M word types corresponding to 9.9B tokens from a large corpus used to pretrain the CAMeLBERT-MSA model (Inoue et al., 2021). Second is *CA-CB*, a set of 2.4M word types corresponding to 0.7B tokens from the CAMeLBERT-CA model (Inoue et al., 2021). Both *MSA-CB* and *CA-CB* were selected to contain Arabic characters only. And finally, *PATB-Train*, the *training* portion of the PATB parts 1v4.1, 2v3.1 and 3v3.2 (Maamouri et al., 2004,Maamouri et al., 2010a,b, 2011) following Diab et al. (2013)'s splits.

Speed Currently, it takes about 45 minutes[2] to generate our DB as part of the **offline DB Maker** process. We also conducted controlled experiments measuring the effect of factorization (see §3) on online speed and found it to make analysis 10% faster. This is due to the fact that the number of categories and compatibility entries is reduced by more than 5.6 times in the factorized version.

We measure the **online analysis** speed of CAMELMORPH MSA DB and SAMA/CALIMA DB (in CPU time) over *MSA-CB* and *CA-CB*. Results in Table 3 show that CAMELMORPH MSA is between 2.4 and 2.9 times slower than SAMA/CALIMA.

Coverage We compare the out-of-vocabulary (OOV) rate of CAMELMORPH MSA DB and SAMA/CALIMA DB over *MSA-CB* and *CA-CB*. Results are in Table 3. Compared to SAMA/CALIMA, CAMELMORPH MSA consistently reduces OOV by 10% (types) and 36% (tokens) for MSA, and by 20% (types) and 38% (tokens) for CA. Manual examination of OOV types shows that proper nouns seem to carry the bigger share of OOVs. Other major sources of OOV include spelling errors, spelling variations of the same word, and erroneous merging of words. These results show that paradigm gap filling, the addition of rare phenomena (especially the energetic mood and indirect object pronominal clitics), and the seamless integration of Wikidata entities had a considerable impact on OOV reduction. Finally, we report that the number of analyses per type in CAMELMORPH MSA is 1.4 times that in SAMA/CALIMA.

[2]On one core of a Mac Book Air M2 with 16GB RAM.

	Camel Morph MSA		SAMA/CALIMA	
	MSA	CA	MSA	CA
Run Time (sec)	12,293	4,667	**4,231**	**1,960**
Type OOV	**67.9%**	**34.7%**	75.1%	43.2%
Token OOV	**2.3%**	**1.5%**	3.5%	2.5%
Analyses/Type	**18.9**	**21.2**	13.7	15.2
Analyses/Token	**38.6**	**45.7**	18.9	20.3

Table 3: Results comparing speed and coverage of CAMELMORPH MSA and SAMA/CALIMA over a large set of Arabic words from MSA (9.9B tokens, 11.4M types) and CA (0.7B tokens, 2.4M types).

Accuracy We assess the quality of CAMELMORPH MSA by evaluating its coverage of *PATB-Train*. We drop all incomplete PATB gold analyses marked with placeholder values (∼1% of all entries). Of the rest, we are able to recall 95.9% (94.5% in unique type space) based on matching on all of lemma, diacritization, and morphological analysis. A human evaluation on a sample of the mismatching instances shows that about 90% of mismatches are actually due to a defect in the gold data. Cases include spelling inconsistencies between lemma and stem, attributing a stem to a wrong lemma because of paradigm ambiguity, or simply wrong analysis on one or more features. Our system handles these cases correctly.

Project Updates Based on the error analyses we conducted above, we updated our CAMELMORPH MSA DB for completeness. We tried to include as many missing phenomena as possible, without compromising our morphological and orthographic guidelines. All updates and future additions will be publicly available.[1]

6. Conclusions and Future Work

We presented CAMELMORPH MSA, the largest open-source Arabic morphological analyzer and generator to date and evaluated it against a popular morphological analyzer, showing superior performance.

In the future, we plan to continue expanding CAMELMORPH MSA to increase its coverage further to include more MSA and CA lexical items, as well as expanding it to include Arabic dialects. We plan to continue filling any gaps in the DB, such as any incomplete diacritizations, transcriptions and missing glosses. We also plan to increase its robustness to input spelling errors and develop smart back-off utilities to allow it to propose plausible answers for unseen words. We also plan on improving speed performance and offering users options for controlled reduced DB sizes.

Acknowledgements

We thank Djellel Difallah for help with Wikidata, and Ossama Obeid for help with Camel Tools.

Ethics Statement

- All annotators have been paid fair wages as part of the work on developing and quality checking the lexical resources and debugging the overall system.
- We acknowledge that our lexical resources may have some unforeseen errors, but expect these to be of minimal risk to downstream applications.
- We also acknowledge that our tool, like many others in NLP, can be used, in the wrong hands, to manipulate texts for harmful purposes.

7. Bibliographical References

Imad A. Al-Sughaiyer and Ibrahim A. Al-Kharashi. 2004. Arabic morphological analysis techniques: A comprehensive survey. *Journal of the American Society for Information Science and Technology*, 55(3):189–213.

Bashar Alhafni, Nizar Habash, and Houda Bouamor. 2022. User-centric gender rewriting. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 618–631, Seattle, United States. Association for Computational Linguistics.

Bashar Alhafni, Go Inoue, Christian Khairallah, and Nizar Habash. 2023. Advancements in Arabic grammatical error detection and correction: An empirical investigation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6430–6448, Singapore. Association for Computational Linguistics.

Ameerah Alothman and Abdul Malik Alsalman. 2020. Arabic morphological analysis techniques: A survey and classification. *International Journal of Advanced Computer Science and Applications*, pages 214–222. Publisher Copyright: © Science and Information Organization.

Mohamed Altantawy, Nizar Habash, and Owen Rambow. 2011. Fast Yet Rich Morphological Analysis. In *Proceedings of the International Workshop on Finite-State Methods and Natural Language Processing (FSMNLP)*, Blois, France.

Ramy Baly, Hazem Hajj, Nizar Habash, Khaled Bashir Shaban, and Wassim El-Hajj. 2017. A sentiment treebank and morphologically enriched recursive deep models for effective sentiment analysis in arabic. *ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP)*, 16(4):23.

Kenneth Beesley. 1996. Arabic Finite-State Morphological Analysis and Generation. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pages 89–94, Copenhagen, Denmark.

Kenneth Beesley, Tim Buckwalter, and Stuart Newton. 1989. Two-Level Finite-State Analysis of Arabic Morphology. In *Proceedings of the Seminar on Bilingual Computing in Arabic and English*.

Mohamed Boudchiche, Azzeddine Mazroui, Mohamed Ould Abdallahi Ould Bebah, Abdelhak Lakhouaja, and Abderrahim Boudlal. 2017. AlKhalil Morpho Sys 2: A robust Arabic morpho-syntactic analyzer. *Journal of King Saud University - Computer and Information Sciences*, 29(2):141–146.

Mona Diab, Nizar Habash, Owen Rambow, and Ryan Roth. 2013. LDC Arabic treebanks and associated corpora: Data divisions manual. *arXiv preprint arXiv:1309.5652*.

Nizar Habash, Salam Khalifa, Fadhl Eryani, Owen Rambow, Dana Abdulrahim, Alexander Erdmann, Reem Faraj, Wajdi Zaghouani, Houda Bouamor, Nasser Zalmout, Sara Hassan, Faisal Al shargi, Sakhar Alkhereyf, Basma Abdulkareem, Ramy Eskander, Mohammad Salameh, and Hind Saddiki. 2018. Unified Guidelines and Resources for Arabic Dialect Orthography. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan.

Nizar Habash, Reham Marzouk, Christian Khairallah, and Salam Khalifa. 2022. Morphotactic modeling in an open-source multi-dialectal Arabic morphological analyzer and generator. In *Proceedings of the 19th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 92–102, Seattle, Washington. Association for Computational Linguistics.

Nizar Habash and Owen Rambow. 2006. MAGEAD: A morphological analyzer and generator for the Arabic dialects. In *Proceedings of the*

International Conference on Computational Linguistics and the Conference of the Association for Computational Linguistics (COLING-ACL), pages 681–688, Sydney, Australia.

Nizar Habash, Abdelhadi Soudi, and Tim Buckwalter. 2007. On Arabic Transliteration. In A. van den Bosch and A. Soudi, editors, *Arabic Computational Morphology: Knowledge-based and Empirical Methods*, pages 15–22. Springer, Netherlands.

Nizar Y Habash. 2010. *Introduction to Arabic natural language processing*, volume 3. Morgan & Claypool Publishers.

Go Inoue, Bashar Alhafni, Nurpeiis Baimukan, Houda Bouamor, and Nizar Habash. 2021. The interplay of variant, size, and task type in Arabic pre-trained language models. In *Proceedings of the Sixth Arabic Natural Language Processing Workshop*, pages 92–104, Kyiv, Ukraine (Virtual). Association for Computational Linguistics.

Go Inoue, Salam Khalifa, and Nizar Habash. 2022. Morphosyntactic tagging with pre-trained language models for Arabic and its dialects. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1708–1719, Dublin, Ireland. Association for Computational Linguistics.

Christian Khairallah, Reham Marzouk, Salam Khalifa, Mayar Nassar, and Nizar Habash. 2024. Computational morphology and lexicography modeling of Modern Standard Arabic nominals. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 1071–1084, St. Julian's, Malta. Association for Computational Linguistics.

George Kiraz. 1994. Multi-tape Two-level Morphology: A Case study in Semitic Non-Linear Morphology. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pages 180–186, Kyoto, Japan.

Mohamed Maamouri, Ann Bies, Tim Buckwalter, and Wigdan Mekki. 2004. The Penn Arabic Treebank: Building a Large-Scale Annotated Arabic Corpus. In *Proceedings of the International Conference on Arabic Language Resources and Tools*, pages 102–109, Cairo, Egypt.

Mohamed Maamouri, Dave Graff, Basma Bouziri, Sondos Krouna, Ann Bies, and Seth Kulick. 2010. Standard Arabic morphological analyzer (sama) version 3.1. *Linguistic Data Consortium, Catalog No.: LDC2010L01*.

Ossama Obeid, Nasser Zalmout, Salam Khalifa, Dima Taji, Mai Oudah, Bashar Alhafni, Go Inoue, Fadhl Eryani, Alexander Erdmann, and Nizar Habash. 2020. CAMeL tools: An open source python toolkit for Arabic natural language processing. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 7022–7032, Marseille, France. European Language Resources Association.

Otakar Smrž. 2007a. ElixirFM — Implementation of Functional Arabic Morphology. In *Proceedings of the Workshop on Computational Approaches to Semitic Languages (CASL)*, pages 1–8, Prague, Czech Republic. ACL.

Otakar Smrž. 2007b. *Functional Arabic Morphology. Formal System and Implementation*. Ph.D. thesis, Charles University in Prague, Prague, Czech Republic.

Ridouane Tachicart, Karim Bouzoubaa, Salima Harrat, and Kamel Smaïli. 2022. *Arabic Dialects Morphological Analyzers: A Survey*, pages 189–203. Springer International Publishing, Cham.

Dima Taji, Salam Khalifa, Ossama Obeid, Fadhl Eryani, and Nizar Habash. 2018. An Arabic Morphological Analyzer and Generator with Copious Features. In *Proceedings of the Fifteenth Workshop on Computational Research in Phonetics, Phonology, and Morphology (SIGMORPHON)*, pages 140–150.

Nasser Zalmout and Nizar Habash. 2017. Don't throw those morphological analyzers away just yet: Neural morphological disambiguation for Arabic. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 704–713, Copenhagen, Denmark.

8. Language Resource References

Tim Buckwalter. 2004. *Buckwalter Arabic Morphological Analyzer Version 2.0*. LDC. Linguistic Data Consortium, Catalog No.: LDC2004L02, ISLRN 694-194-540-336-4.

Mohamed Maamouri, Ann Bies, Seth Kulick, Fatma Gaddeche, Wigdan Mekki, Sondos Krouna, Basma Bouziri, and Wadji Zaghouani. 2010a. Arabic treebank: Part 1 v 4.1. Linguistic Data Consortium (LDC2010T13).

Mohamed Maamouri, Ann Bies, Seth Kulick, Fatma Gaddeche, Wigdan Mekki, Sondos Krouna, Basma Bouziri, and Wadji Zaghouani. 2011. Arabic treebank: Part 2 v 3.1. Linguistic Data Consortium (LDC2011T09).

Mohamed Maamouri, Ann Bies, Seth Kulick, Sondos Krouna, Fatma Gaddeche, and Wajdi Zaghouani. 2010b. Arabic treebank: Part 3 v 3.2. Linguistic Data Consortium (LDC2010T08).

Maamouri, Mohamed and Graff, Dave and Bouziri, Basma and Krouna, Sondos and Bies, Ann and Kulick, Seth. 2010c. *Standard Arabic morphological analyzer (SAMA) version 3.1*. LDC. Linguistic Data Consortium, Catalog No.: LDC2010L01, ISLRN 898-935-705-624-6.

A. Sample Specifications

Figure 2 shows a small sample of specifications for three Arabic verbs, and a very small chunk of their inflectional paradigms. Some specifications like the passive forms and transcriptions are generated by scripts which are provided as input to the DB Maker algorithm (see Figure 1).

I. **ORDER**

	PREFIX	STEM	SUFFIX
(a)	[QUES] [CONJ]	[STEM-PV]	[PVSuff] [PRON]

II. **MORPH**

	CLASS	FUNC	FORM	BW	GLOSS	FEAT	COND-REQ	COND-SET
(b)	[QUES]					prc3:0		
(c)	[QUES]	>a/PART_INTERROG	>a	>a/INTERROG_PART	is/are	prc3:>a_ques		
(d)	[CONJ]					prc2:0		
(e)	[CONJ]	fa/CONJ	fa	fa/CONJ	so/and	prc2:fa_conj		
(f)	[CONJ]	wa/CONJ	wa	wa/CONJ	and	prc2:wa_conj		
(g)	[PVSuff]	PVSuff.2MS	ta	ta/PVSUFF_SUBJ:2MS	you_[m.s.]	asp:p per:2 gen:m num:s	else	c-suff
(h)	[PVSuff]	PVSuff.2MS	~a	~a/PVSUFF_SUBJ:2MS	you_[m.s.]	asp:p per:2 gen:m num:s	#t	c-suff
(i)	[PVSuff]	PVSuff.2FS	ti	ti/PVSUFF_SUBJ:2FS	you_[f.s.]	asp:p per:2 gen:f num:s	else	c-suff #^i
(j)	[PVSuff]	PVSuff.2FS	~i	~i/PVSUFF_SUBJ:2FS	you_[f.s.]	asp:p per:2 gen:f num:s	#t	c-suff #^i
(k)	[PVSuff]	PVSuff.3MS	a	a/PVSUFF_SUBJ:3MS	he	asp:p per:3 gen:m num:s	else	v-suff
(l)	[PVSuff]	PVSuff.3FS	at	at/PVSUFF_SUBJ:3FS	she	asp:p per:3 gen:f num:s	else	v-suff
(m)	[PRON]					enc0:0		
(n)	[PRON]	hu/PRON.3MS	hu	hu/XVSUFF_DO:3MS	him	enc0:3ms_dobj	trans else	enc0
(o)	[PRON]	hu/PRON.3MS	hi	hu/XVSUFF_DO:3MS	him	enc0:3ms_dobj	trans #^i	enc0
(p)	[PRON]	hA/PRON.3FS	hA	hA/XVSUFF_DO:3FS	her	enc0:3fs_dobj	trans	enc0

III. **LEXICON**

	CLASS	LEMMA	FORM	BW	GLOSS	FEAT	COND-REQ	COND-SET
(q)	[STEM-PV]	katab	katab	katab/PV	write	pos:verb asp:p		trans
(r)	[STEM-PV]	Saw~at	Saw~at	Saw~at/PV	vote	pos:verb asp:p		#t trans
(s)	[STEM-PV]	fAt	fAt	fAt/PV	go_by	pos:verb asp:p	v-suff	#t trans
(t)	[STEM-PV]	fAt	fut	fut/PV	go_by	pos:verb asp:p	c-suff	#t trans

Figure 2: Sample Morphological Specifications for MSA perfective verbs, as they would appear in the specification sheets. **COND-REQ** and **COND-SET** refer to **COND-T** and **COND-S**, respectively. Note that the pattern and root information is missing from the **LEXICON** section of this table.