G_i2P_i: Rule-based, index-preserving grapheme-to-phoneme transformations

Aidan Pine[1] aidan.pine
Patrick Littell[1] patrick.littell
Eric Joanis[1] eric.joanis
David Huggins-Daines[2] dhdaines@gmail.com
Christopher Cox[3] christopher.cox
Fineen Davis[4] fineen.davis@gmail.com
Eddie Antonio Santos[5] eddie.santos@ucdconnect.ie
Shankhalika Srikanth[6] ssrikanth@uvic.ca
Delasie Torkornoo[3] delasie.torkornoo
Sabrina Yu[7] sab.yu@mail.utoronto.ca

Abstract

This paper describes the motivation and implementation details for a rule-based, index-preserving grapheme-to-phoneme engine 'G_i2P_i' implemented in pure Python and released under the open source MIT license[8]. The engine and interface have been designed to prioritize the developer experience of potential contributors without requiring a high level of programming knowledge. G_i2P_i already provides mappings for 30 (mostly Indigenous) languages, and the package is accompanied by a web-based interactive development environment, a RESTful API, and extensive documentation to encourage the addition of more mappings in the future. Finally, we discuss three downstream applications of G_i2P_i, and show results of preliminary evaluation.

1 Introduction and motivation

G_i2P_i is a library[9] for grapheme-to-phoneme and orthographic transformation, with a particular focus on the needs of digital humanities projects. While libraries for general-purpose G2P exist, we found that our downstream projects had special needs that existing libraries did not entirely meet. In particular,

1. Subject-matter experts for these languages are often teachers or linguists without a background in computer science, who are unfamiliar with the conventions of programming and need more intuitive interfaces to convert their knowledge into executable code (§2.1).

2. Most existing libraries operate on unstructured text, under the assumption that the original document will be discarded after its linguistic information is extracted. Our downstream use-cases, however, often involve the augmentation of the original document with downstream results (e.g., with pronunciations, alternative orthographies, or time-aligned highlighting). We need to be able to trace results backward to their original counterparts (e.g. by using indices as shown in Figure 1), maintaining this information through every step of transduction, so that markup, IDs, punctuation, and other features of the original document can be preserved (§2.2).

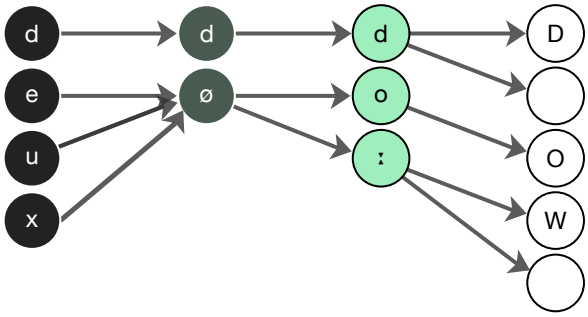


Figure 1: Screenshot from G2P Studio of interactive visualization of the indices preserved when composing transductions of the French word "deux", between the orthographic form, a phonetic representation in the International Phonetic Alphabet (IPA), the closest English phonemes according to `PanPhon` (Mortensen et al., 2016), and finally to English ARPABET (see §2.3).

3. Software packages for linguistic transformation, and their dependencies, can be difficult to compile and install, or cannot be installed on all operating systems.

The need for such specialized knowledge presents a bottleneck in the development of G2P

[1]National Research Council Canada; `@nrc-cnrc.gc.ca`
[2]Independent Researcher
[3]Carleton University; `@carleton.ca`
[4]Wiichihitotaak ILR Inc.
[5]University College Dublin
[6]University of Victoria
[7]University of Toronto
[8]https://github.com/roedoejet/g2p
[9]The package is registered on the Python Package Index as 'g2p' - however to disambiguate our package from the generic NLP task 'G2P', we make specific reference to the index preservation capabilities of our package and refer to the package as G_i2P_i throughout this paper.

engines, particularly when we venture away from the NLP space and into the digital humanities space; experts of a particular language's sound patterns should not necessarily need experience in programming and compiling software in order to translate their knowledge of the language into a machine-readable format.

Meanwhile, however, the languages that we are concentrating on (in particular, Indigenous languages spoken in Canada) do not typically have extensive, publicly-available corpora of parallel orthographic and phonetic renderings, from which we could learn a weighted FST (Novak et al., 2016; Deri and Knight, 2016) or neural model (Rao et al., 2015; Peters et al., 2017). For most of these languages, rule-based approaches based on expert knowledge will be the norm for the foreseeable future. Fortunately, these are mostly languages with regular, linguistically-informed orthographies, such that rule-based approaches are adequate.

In broad strokes, our library is most similar to Epitran (Mortensen et al., 2018), which shares some of these design decisions; it prioritizes ease of installation and adopts a method for defining rule-based G2P mappings inspired by phonological re-write rule syntax that would be familiar to linguists. Our work differs by allowing rules to be written in a spreadsheet format (§2.1), by having a core engine that preserves the indices between inputs and output transductions (§2.2), and by providing a bundled web interface for writing and running G2P mappings (§2.4) with an accompanying RESTful API (§2.6.1).

2 G_i2P_i

This section briefly describes the process for writing rules (§2.1), the motivation and implementation details for preserving indices between inputs and outputs (§2.2), the automatic generation of cross-linguistic phoneme-to-phoneme mappings (§2.3), the 'G2P Studio' development environment (§2.4), a list of currently supported languages (§2.5), documentation information (§2.6), and a description of various applications (§2.7).

2.1 Writing Rules

Rules are written in either a tabular, spreadsheet format or in JSON (See Figure 2). The core functionality of G_i2P_i is expressible in the spreadsheet format (CSV), while the JSON format allows for more functionality. Each mapping is also accompanied by a configuration file written in YAML. In its most basic form, a rule just has an input and an output, like in Figure 2.

```
a,b
```

(a) Minimal CSV Rule

```
{
    "in": "a",
    "out" : "b"
}
```

(b) Minimal JSON Rule

Figure 2: A minimal rule converting 'a' to 'b' expressed in both the CSV syntax (a) and JSON syntax (b)

Context-sensitive rules can also be written which conditionally apply rules based on whether a pattern is matched before or after the input as shown in Figure 3.

```
{
    "in": "a",
    "out" : "b",
    "context_before": "b",
    "context_after": "c"
}
```

Figure 3: A minimal context-sensitive rule in JSON format for converting 'a' to 'b' only when 'a' is preceded by 'b' and followed by 'c'. The equivalent rule written in the CSV format is '`a,b,b,c`'.

Under the hood, these rules are compiled into regular expressions, where the input is the pattern to match, and the 'context before' and 'context after' values are turned into positive lookbehinds and lookaheads respectively. Lookbehinds are first converted to be fixed width and several other preprocessing steps are applied before constructing the regular expression. Namely, any explicit indices (§2.2) are removed, optional case insensitivity flags are applied, Unicode normalization (NFC or NFD) is done, and special characters can be escaped.

A collection of rules with a configuration constitutes a 'mapping' which can then be run in the sequence the rules are defined or in an automatically generated order that runs the rules in reverse order of input length. This mode is intended to help prevent particular rule 'bleeding' relationships where if the input to a hypothetical rule r_1 is a substring

(1)	*baata*	(2)	*baata*
	r_2 bæta		r_1 bəətə
	r_1 bætə		r_2
	bætə		bəətə

Figure 4: Example of rule ordering relationships in a made-up language. r_1 is the rule a → ə, and r_2 is the rule aa → æ. In this made-up language, 'bætə' is the correct transduced form of 'baata', and therefore we want to order rule r_2 before r_1, as shown in (1), so that r_1 does not bleed the context for r_2 to apply, as shown in (2).

of the input to rule r_2 and is ordered to apply first, it will remove, or 'bleed' the context for r_2 to apply, erroneously preventing the application of r_2 as shown in example (2) in Figure 4.

2.1.1 Preventing Feeding Relationships

Another type of rule interaction that can be avoided is a feeding relationship between rules—i.e. where the output of one rule creates the context for another rule to apply. In some situations this is desired, but it can also create problems in your rules. To handle this, we allow `prevent_feeding` to be declared either for an individual rule in a JSON-formatted mapping or for each rule in a mapping. When `prevent_feeding` is set to `true`, the output of a rule is replaced with a character from the Supplementary Private Use Area A Unicode block, offset by the index of the rule in a given mapping. Thus, they will never match the input or context of other rules. After applying all rules in a mapping, these intermediate representations are transformed back into the appropriate values.

2.1.2 Composite Transducers

In practice, many real-world transduction tasks comprise a sequence of simpler transductions. For example, a G2P transduction used in ReadAlong Studio (§2.7.3) might start with converting a font-encoded orthography into a Unicode compliant form, then replacing confusable characters, converting the orthographic form into IPA, mapping those characters onto their closest English equivalents, and finally mapping the English IPA characters into the ARPABET alphabet used by the acoustic model.

G_i2P_i is built with this in mind; an arbitrary number of transducers can be combined, and chains of transducers can be inferred automatically. If the user requests a mapping from one language code[10] to another, e.g., from `alq` (Algonquin) to `eng-arpabet`, and that particular mapping does not exist, the software can search for the shortest possible chain of transducers with those endpoints, and it will act as if it were an ordinary transducer (including maintaining indices between the ultimate inputs and outputs, and all intermediate forms, as seen in Fig. 1).

Fig. 5 on the following page illustrates the current network of transducers possible in G_i2P_i.

This modularity is intended, in part, to help subject-matter experts contribute their domain knowledge (e.g., the pronunciation of their language's orthography) without having to understand the other specialized components of the transduction pipeline (e.g., confusable Unicode characters or ARPABET), or even the structure of the pipeline as a whole. They only have to contribute their particular piece; G_i2P_i can compose the pipeline as a whole, and even auto-generate certain kinds of missing pieces (§2.3).

2.1.3 Debugging

Debugging transductions can be a difficult task when there are multiple mappings involved, each with possibly dozens of rules. In order to help ease the burden on developers, multiple debugging tools have been developed to assist contributors. In the G2P Studio (§2.4), there is an automatic visualization mode which allows users to visualize transductions in an interactive way; Figure 1 is a screenshot of this visualization.

There are also two alternative options for debugging; the `--debugger` flag used in either the CLI or REST API shows each transduction applied in sequence along with any intermediate steps (§2.1.1). Additionally, there is a `g2p doctor` command in the CLI that checks a specific mapping for a list of common errors, such as the declaration of IPA characters not recognized by PanPhon.

2.2 Preserving Indices

The concerns in (§1) are not as pressing when considering a G2P transformation to create, say, training data for a speech recognition model. There,

[10] In G_i2P_i, a mapping is a collection of rules with a defined input language code and output language code. By "code" we mean an arbitrary label for the language. By convention we start with the ISO 639-3 code and add a descriptive suffix (e.g. `-ipa` when required.

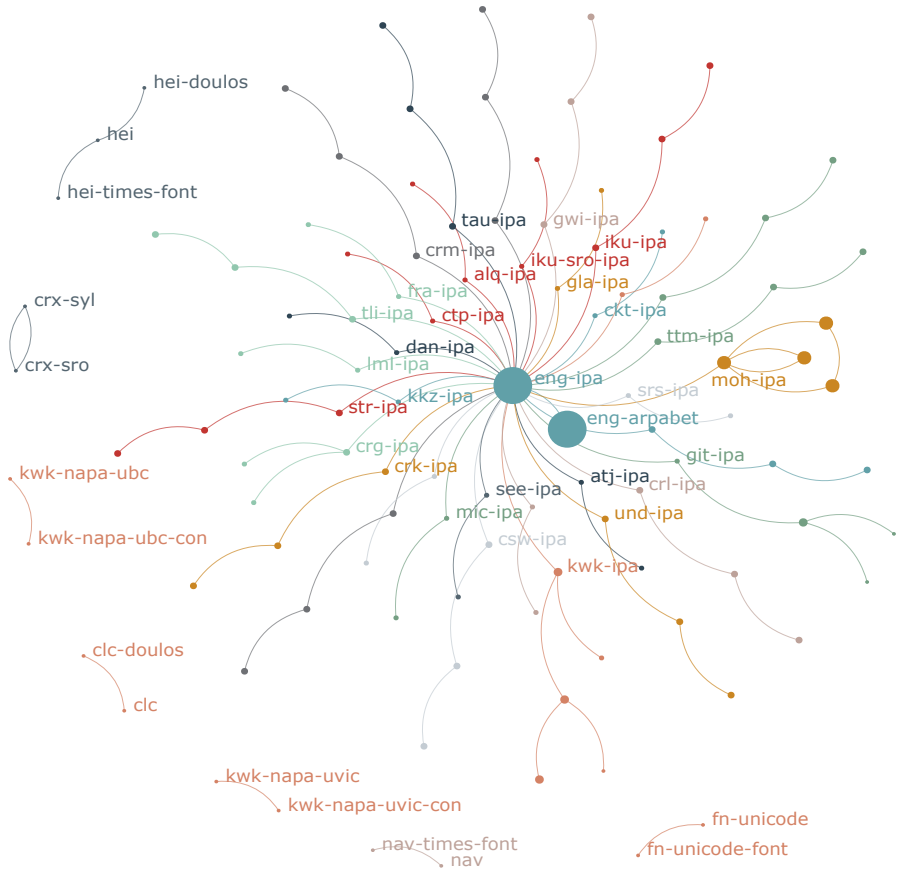


Figure 5: Visualization of the network created by G2P. Nodes represent orthographies or phonetic representations. They are labelled and colour coded according to the associated language's ISO 639-3 code. Arcs represent mappings. Nodes are sized relative to the number of upstream nodes. English (eng) has the largest nodes due to the large number of generated mappings into English for the purpose of the ReadAlong Studio project (see §2.3, §2.7.3).

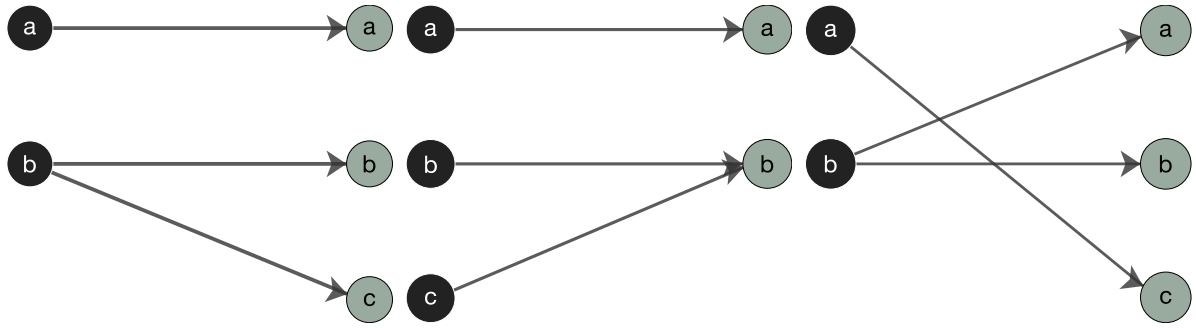
features in the original document like punctuation are ephemera; once the necessary information has been extracted from the document it is henceforth ignored, and only the transformed version is used.

However, consider how the project needs differ when force-aligning a storybook with a recording of it, so that a beginner reader can see words highlighted when they are read, click to hear words in isolation, etc. If our transformation pipeline has thrown out all non-speech features of the document on the way to the ARPABET needed by the decoder, we are left with timestamps that correspond only to a text document with an unclear relationship with the original structured data. This would be fine if the storybook were only to be used as training data, but if we want to re-associate those timestamps with the original document, we would have an additional problem of re-alignment.

We could potentially try to learn an alignment model between the output and the original document, but data is extremely scarce in most of our target languages, and in any case these alignments are something that the model itself could have maintained. Therefore, we designed the G_i2P_i library to maintain index alignments throughout the process, even when transformations are composed.

This is also true below the level of the word. Many of our target languages are very morphologically complex and long words are the norm. Therefore, educational material in these languages often has subword highlighting. For example, educational material from the Onkwawenna Kentyohkwa immersion school for the Kanyen'kéha (Mohawk) language has a systematic association with particular kinds of morphemes with colors. Another example is when the downstream project involves subword phenomena: a bouncing-ball sing-along video requires syllable-level alignment to get the bounce at the correct place and time.

However, in both of these examples, the unit of transformation is still the word; the word is the domain over which most phonological transformations apply. Splitting the original word into subword units before processing will not necessarily produce correct results, as this would introduce new "word" boundaries. Therefore, we must also keep and compose indices below the level of the word, so that even when transforming whole words we can associate the resulting pronunciations, timestamps, etc. with subword markup in the original



(a) ab → abc (b) abc → ab (c) a{1}b{2} → ab{2}c{1}

Figure 6: Examples of various strategies for assigning indices between inputs and outputs; default assignment of indices is shown in 6a and 6b and explicit assignment of indices is shown in 6c.

document.

Maintaining these indices is also useful for a debugging visualization in the bundled development environment (§2.4), making clear in composed transductions how inputs, intermediate, and output forms correspond to each other (Fig. 1).

2.2.1 Default and Explicit Indexing

The default interpretation of rules is to assign indices between inputs and outputs evenly; if there is a mismatch in length between inputs and outputs, excess characters are assigned the index of the last character of the shorter string as seen in Figures 6a and 6b.

However, G_i2P_i also allows a more explicit syntax for defining indexing relationships between inputs and outputs: rules can be marked up with curly braces to indicate a specific indexing of characters between inputs and outputs as seen in Figure 6c.

2.3 Automatic Phoneme-to-Phoneme Mappings

Another use of the G_i2P_i library, beyond grapheme-to-phoneme transformation or orthographic transliteration, is to map the sounds of one language onto the sounds of another, for cross-linguistic comparison. This is used, for example, in ReadAlong Studio (§2.7.3) to align text and speech in an arbitrary language using only an English-language acoustic model.

While these mappings can be written by hand, it is somewhat of a specialized art, typically performed by speech technology specialists. Therefore, the G_i2P_i library also includes functionality to automatically generate phone-to-phone mappings, by leveraging the phone-to-phone distance metrics included in PanPhon (Mortensen et al., 2016), to serve as "glue" in a composite transducer (§2.1.2).

For composing a mapping A with a mapping B, where both the output vocabulary of A and in the input vocabulary B represent IPA characters (but not necessarily the same inventory of IPA characters), the G_i2P_i library can generate a mapping in which each character in the output of A is mapped to its nearest neighbor in the input of B, according to the PanPhon's calculated phone-to-phone distance between the characters' phonological feature vector representations. PanPhon allows for a variety of distance metrics between IPA characters, by default we use PanPhon's Hamming distance metric between IPA phonological feature vector representations. This allows non-specialist users to generate cross-linguistic transductions of the sort used in cross-lingual speech synthesis (§2.7.2) or ReadAlong Studio (§2.7.3), without necessarily having to be a linguist familiar with the IPA.

2.4 G2P Studio

In addition to developing rules locally as described in §2.1, writing and running mappings can be performed in a web interface called 'G2P Studio'. The G2P Studio is written using a vanilla JavaScript front-end with Skeleton CSS[11] and a Python back-end written in Flask with low-latency, bidirectional communication handled through WebSockets.

The G2P Studio is hosted at `https://bit.ly/g2p-studio` but can also be deployed in a distributed fashion, as the lightweight server/app code is bundled in the Python package.

2.4.1 Visual Programming Rule Creator

In addition to creating rules in spreadsheets or JSON files, G2P Studio includes a visual program-

[11] `http://getskeleton.com/`

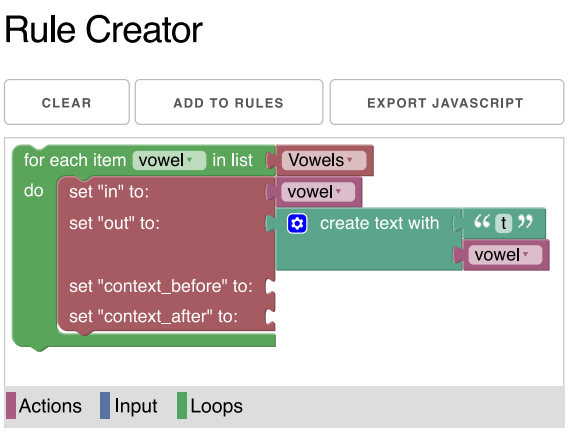


Figure 7: Screenshot of the "Rule Creator" interface in G2P Studio showing a toy set of rules being made that take each vowel in the Vowels variable (declared elsewhere in G2P Studio) as input and return the same vowel prefixed by 't' as output.

ming interface for authoring rules (Figure 7). This visual programming interface was created with Blockly[12] (Fraser, 2015; Pasternak et al., 2017).

2.5 Supported Languages

At the time of writing, 30 languages are supported: Anishinàbemiwin (alq), Atikamekw (atj), Michif (crg), Southern & Northern East Cree (crj), Plains Cree (crk), Moose Cree (crm), Swampy Cree (csw), Western Highland Chatino (ctp), Danish (dan), French (fra), Gitksan (git), Scottish Gaelic (gla), Gwich'in (gwi), Hän (haa), Inuinnaqtun (ikt), Inuktitut (iku), Kaska (kkz), Kwak'wala (kwk), Raga (lml), Mi'kmaq (mic), Kanien'kéha (moh), Anishinaabemowin (oji), Seneca (see), Tsuut'ina (srs), SENĆOŦEN (str), Upper Tanana (tau), Southern Tutchone (tce), Northern Tutchone (ttm), Tagish (tgx), Tlingit (tli). G$_i$2P$_i$ is also bundled with other mappings, such as mappings from font-encoded writing systems in Heiltsuk, Tsilqot'in, and Navajo to Unicode-compliant versions as well a mapping from English IPA to English ARPABET.

2.6 Documentation

Documentation on primary use cases and edge cases is an important part of the G$_i$2P$_i$ project. Without contributions to the mappings, the project will be less accessible, and more difficult to maintain. Technical documentation is therefore provided through ReadTheDocs[13] as well as a 7-part blog series[14] written for a more general audience.

2.6.1 RESTful API documentation

The core functionality of G$_i$2P$_i$ is also exposed through a RESTful API. The API and its documentation are generated dynamically using Swagger[15] to provide up-to-date, interactive documentation[16]. The documentation allows users to interactively make requests to the API, see available mappings, and copy the related Curl commands along with other information like the Request URLs, and Response body and headers from their requests.

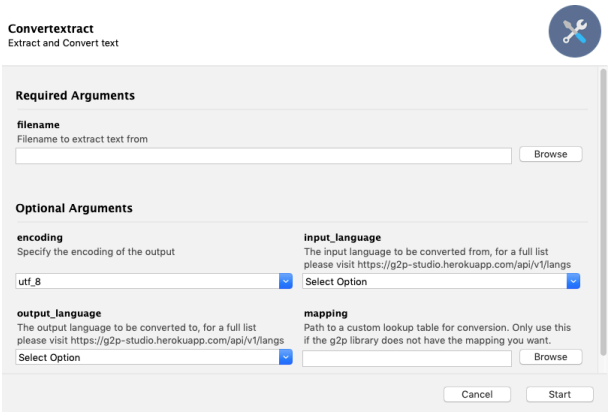


Figure 8: Screenshot of Convertextract GUI for macOS

2.7 Applications

Grapheme-to-phoneme transformations are used in a wide variety of natural language processing tasks, and so G$_i$2P$_i$ can be used for any such use case. Below, we briefly discuss three projects that are implemented using G$_i$2P$_i$.

2.7.1 Convertextract

Convertextract (Pine and Turin, 2018), is a tool that performs find/replace operations on Microsoft Office documents while preserving the original formatting of the file. Convertextract is available for the command line and with a macOS GUI (Figure 8). Integration is fully automated between the libraries—whenever a new version of G$_i$2P$_i$ is released, it triggers a new build and release of convertextract which is able to perform any conversion between any of the mappings defined in G$_i$2P$_i$.

2.7.2 Speech Synthesis Front End

We have built speech synthesis models for SENĆOŦEN, Kanyen'kéha, and Gitksan using

[12] https://developers.google.com/blockly
[13] https://g2p.readthedocs.io/
[14] https://blog.mothertongues.org/g2p-background/
[15] https://swagger.io/
[16] https://bit.ly/g2p-api-docs

mappings developed with G_i2P_i (Pine et al., 2022). Part of the pipeline for these models involves transforming the orthographic form of utterances to a phonetic representation. This is necessary for the pre-processing step of forced alignment and the phonetic representation of the text is used as input to the feature prediction network in the speech synthesis pipeline. The phonetic form is represented either as one-hot encodings or as multi-hot phonological feature vectors derived from PanPhon. This is another use case for generated mappings (§2.3); a mapping between the IPA symbols of a target language could be mapped on to the IPA symbols of one or more languages in a pre-trained model using G_i2P_i to facilitate fine-tuning on a language that was not present in the pre-trained model. This method allows for a principled rule-based method for mapping between symbol spaces in cross-lingual speech synthesis without the use of a learned phonetic transformation network like the one described by Tu et al. (2019).

2.7.3 ReadAlong Studio

ReadAlong Studio[17] is a library for the creation of time-aligned "read-along" audiobooks, intended to make text/speech alignment easy for non-specialist users. It utilizes a zero-shot speech alignment paradigm in which target-language text is converted to English-language phonemes, and then force-aligned using an English-language acoustic model (specifically, the model from Huggins-Daines et al. (2006)).

By its nature, this text conversion is a composite transduction (§2.1.2) – first converting the target-language text to target-language phonemes, then converting the target-language phonemes into similar English-language phonemes (§2.3), and finally converting the English-language phonemes into the ARPABET symbols that the aligner expects as shown previously in Figure 1.

While none of these steps is, in itself, difficult to specify by hand, in combination they require a relatively rare expertise: (1) understanding of a specific language's orthography, (2) understanding how sounds map to each other between languages, and (3) familiarity with the ARPABET conventions and the specific phone vocabulary of the English-language acoustic model used.

By automating the second and third steps, and automating their composition, the G_i2P_i library only requires the user to be able to do the first, putting it within reach of a linguistically-informed teacher or other knowledge worker. It *does* require knowledge of the IPA, but this is relatively widespread knowledge, and IPA-equivalence charts for many languages are easy to come by in books and online.

3 Evaluation

As mentioned previously, this paper shares many similarities with Epitran, however we cannot evaluate our system using the same method. Epitran leverages baseline data available in some of the languages it supports to evaluate the system indirectly using the downstream task of developing ASR systems. The word error rates (WER) of ASR systems created using letter-to-sound rules from Epitran are then compared against those created using the available baseline. The primary focus for this library is on extremely low resource languages, and we do not possess baseline data to recreate the evaluation procedure implemented by Epitran.

As a crude replacement, we evaluate two of our mappings by reporting the accuracy of a downstream forced alignment task using ReadAlong Studio (§2.7.3). We manually annotated data from SENĆOŦEN and Kanyen'kéha with word-level alignments in Praat. Given the time-consuming nature of manual alignment, we were limited to a single document from each language; the SENĆOŦEN document is 5:47 long and contains 417 words and the Kanyen'kéha document is 5:07 long and contains 249 words. Both documents are private materials owned by the language communities and shared with us by linguist Timothy Montler and Kanyen'kéha educator Owennatekha Brian Maracle respectively.

We evaluate our hand-written mappings against a baseline zero-shot G2P method. The baseline we use is ReadAlong Studio's fallback method for 'und' (ISO 639-3 for 'undetermined') text. This fallback method uses the text-unidecode[18] package to convert all characters to ASCII equivalents, and then uses a rule-based mapping from ASCII to IPA. For our hand-written SENĆOŦEN and Kanyen'kéha mappings, we use G_i2P_i's built-in automatic mapping functionality to map the IPA inventories to the closest English IPA equivalents (§2.3).

Similar to McAuliffe et al. (2017), we evaluate

[17] https://github.com/ReadAlongs/Studio

[18] https://pypi.org/project/text-unidecode/

Mapping	Lang.	Tolerance (ms)			
		<10	<25	<50	<100
Handmade	moh	0.24	0.43	0.68	0.84
	str	**0.24**	**0.49**	**0.69**	**0.88**
Und	moh	0.24	0.46	0.72	0.86
	str	0.15	0.34	0.49	0.62

Table 1: Results for Kanyen'kéha (moh) and SENĆOŦEN (str) downstream forced alignment task showing alignment accuracy with varying amounts of tolerance for word boundaries for alignments created from handmade G_i2P_i mappings and mappings based on text unidecode ('Und'), measured against hand-labelled alignments.

the system by reporting the accuracy of the word boundaries predicted by the aligner within thresholds of < 10, < 25, < 50, and < 100 milliseconds; for example, a result of 0.88 with a threshold of <100ms means that 88% of system boundaries were within 100ms of the reference boundaries.

As shown in Table 1, the results for SENĆOŦEN and Kanyen'kéha are not the same. While alignment created from handmade mappings for SENĆOŦEN outperforms the baseline by 26% with a 100ms tolerance threshold, the results from Kanyen'kéha are less clear, and do not show an improvement over the baseline. We suspect this is in part because while the Kanyen'kéha orthography is quite consistent with other Latin-based orthographies, the SENĆOŦEN orthography is considerably different (for example Ć corresponds to the sound /ʧ/), which would affect the text-unidecode library's ability to predict reasonable ASCII equivalents. These results could point to a finding that for simpler[19] orthographies that are strongly aligned with English, the text unidecode technique could be sufficient; however, caution should be applied in interpreting these preliminary results and further evaluation with additional languages, data, and downstream tasks would be needed.

4 Conclusion

In this paper we presented G_i2P_i along with its motivations, and some descriptions of its use cases. The library is written in pure Python to support (relatively) easy installation, with support for 30 different languages, index preservation between inputs and outputs, an accompanying graphical web interface, a RESTful API, and extensive documentation to encourage the development of mappings for more languages in the future.

We recognize that language experts are the best people suited to write mappings between a language's orthography and the IPA, and we hope that through a variety of features that such a contributor would "get for free" by contributing, that G_i2P_i is an attractive option for rule-based G2P. To summarize, by contributing a mapping, a contributor will acquire the following:

- Integration into the broader G_i2P_i transduction network for cross-lingual purposes (§2.1.2)
- Debugging tools (§2.1.3)
- Index preservation for transductions (§2.2)
- A graphical interface (§2.4)
- A RESTful API (§2.6.1)
- Automatic downstream support in Convertextract (§2.7.1) and ReadAlong Studio (§2.7.3)

Each time a mapping is added to G_i2P_i it becomes more useful software, so we have prioritized the developer experience of contributing a mapping through documentation, debugging tools and the features described above. We hope that these measures will make using and contributing to G_i2P_i more accessible and we will measure the success of the project by the number of collaborator contributions of mappings.

Acknowledgements

We would like to acknowledge the Yukon Native Language Centre and the W̱SÁNEĆ School Board for extremely helpful contributions in providing alphabet charts, sample text and other materials that aided in the development of some of the mappings described in §2.5. We would also like to thank Bradley Ellert for his contributions in helping developing mappings.

References

Aliya Deri and Kevin Knight. 2016. Grapheme-to-phoneme models for (almost) any language. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 399–408.

[19] Kanyen'kéha contains fewer than half as many segmental phonemes as SENĆOŦEN

Neil Fraser. 2015. Ten things we've learned from Blockly. In *2015 IEEE Blocks and Beyond Workshop*, pages 49–50.

David Huggins-Daines, Mohit Kumar, Arthur Chan, Alan W Black, Mosur Ravishankar, and Alexander I Rudnicky. 2006. Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 1, pages I–I. IEEE.

Michael McAuliffe, Michaela Socolof, Sarah Mihuc, Michael Wagner, and Morgan Sonderegger. 2017. Montreal forced aligner: Trainable text-speech alignment using kaldi. In *Interspeech*, volume 2017, pages 498–502.

David R. Mortensen, Siddharth Dalmia, and Patrick Littell. 2018. Epitran: Precision G2P for many languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Paris, France. European Language Resources Association.

David R. Mortensen, Patrick Littell, Akash Bharadwaj, Kartik Goyal, Chris Dyer, and Lori S. Levin. 2016. Panphon: A resource for mapping IPA segments to articulatory feature vectors. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3475–3484. Association for Computational Linguistics.

Josef Robert Novak, Nobuaki Minematsu, and Keikichi Hirose. 2016. Phonetisaurus: Exploring grapheme-to-phoneme conversion with joint n-gram models in the wfst framework. *Natural Language Engineering*, 22(6):907–938.

Eric Pasternak, Rachel Fenichel, and Andrew N. Marshall. 2017. Tips for creating a block language with Blockly. In *2017 IEEE Blocks and Beyond Workshop*, pages 21–24.

Ben Peters, Jon Dehdari, and Josef van Genabith. 2017. Massively Multilingual Neural Grapheme-to-Phoneme Conversion. In *Proceedings of the First Workshop on Building Linguistically Generalizable NLP Systems*, pages 19–26, Copenhagen, Denmark. Association for Computational Linguistics.

Aidan Pine and Mark Turin. 2018. Seeing the Heiltsuk orthography from font encoding through to Unicode: A case study using convertextract. In Claudia Soria, Laurent Besacier, and Laurette Pretorius, editors, *Proceedings of the LREC 2018 Workshop "CCURL 2018 – Sustaining knowledge diversity in the digital age"*, pages 27–30. European Language Resources Association.

Aidan Pine, Dan Wells, Nathan Thanyehténhas Brinklow, Patrick Littell, and Korin Richmond. 2022. Requirements and Motivations for Low Resource Speech Synthesis. In *Proceedings of ACL 2022*, Dublin, Ireland. Association for Computational Linguistics.

Kanishka Rao, Fuchun Peng, Haşim Sak, and Françoise Beaufays. 2015. Grapheme-to-phoneme conversion using long short-term memory recurrent neural networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4225–4229.

Tao Tu, Yuan-Jui Chen, Cheng-chieh Yeh, and Hung-yi Lee. 2019. End-to-end Text-to-speech for Low-resource Languages by Cross-Lingual Transfer Learning. In *Interspeech 2019*, pages 2075–2079.