Lexi: A tool for adaptive, personalized text simplification

Joachim Bingel[1] **Gustavo H. Paetzold**[2] **Anders Søgaard**[1]
[1] Department of Computer Science, University of Copenhagen, Denmark
[2] Federal University of Technology - Paraná, Brazil
{bingel,soegaard}@di.ku.dk, ghpaetzold@utfpr.edu.br

Abstract

Most previous research in text simplification has aimed to develop generic solutions, assuming very homogeneous target audiences with consistent intra-group simplification needs. We argue that this assumption does not hold, and that instead we need to develop simplification systems that adapt to the individual needs of specific users. As a first step towards personalized simplification, we propose a framework for adaptive lexical simplification and introduce Lexi, a free open-source and easily extensible tool for adaptive, personalized text simplification. Lexi is easily installed as a browser extension, enabling easy access to the service for its users.

1 Introduction

Many a research paper on text simplification starts out by sketching the problem of text simplification as rewriting a text such that it becomes easier to read, changing or removing as little of its informational content as possible (Zhu et al., 2010; Coster and Kauchak, 2011; De Belder and Moens, 2010; Paetzold and Specia, 2015; Bingel and Søgaard, 2016). Such a statement may describe the essence of simplification as a research task, but it hides the fact that it is not always easy to decide what is easy for a particular user. This paper discusses why we need custom-tailored simplifications for individual users, and argues that previous research on non-adaptive text simplification has been too generic to unfold the full potential of text simplification.

Even when limiting ourselves to lexical substitution, i.e. the task of reducing the complexity of a document by replacing difficult words with easier-to-read synonyms, we see plenty of evidence that, for instance, dyslexics are highly individual in what material is deemed easy and complex (Ziegler et al., 2008). Lexi, which we introduce in this paper, is a free, open-source and easily extensible tool for adaptively learning what items specific users find difficult, using this information to provide better (lexical) simplification. Our system initially serves Danish, but is easily extended to further languages. For surveys of text simplification, including resources across languages, see Siddharthan (2014), Shardlow (2014b) and Collins-Thompson (2014).

1.1 There is no one-size-fits-all solution to text simplification

Text simplification is a diverse task, or perhaps rather a family of tasks, with a number of different target audiences that different papers and research projects have focused on. Among the most prominent target audiences are foreign language learners, for whom various approaches to simplifying text have been pursued, often focusing on lexical (Tweissi, 1998) but also sentence-level simplification (Liu and Matsumoto, 2016). Other notable groups that have been specifically targeted in text simplification research include dyslexics (Rello et al., 2013), and the aphasic (Carroll et al., 1998), for whom particularly long words and sentences, but also certain surface forms such as specific character combinations, may pose difficulties. People on the autism spectrum have also been addressed, with the focus lying on reducing the amount of figurative expressions in a text or reducing syntactic complexity (Evans et al., 2014). Reading beginners (both children and adults) are another group with very particular needs, and text simplification

Proceedings of the 27th International Conference on Computational Linguistics, pages 245–258
Santa Fe, New Mexico, USA, August 20-26, 2018.

research has tried to provide this group with methods to reduce the amount of high-register language and non-frequent words (De Belder and Moens, 2010).

Evidently, each target group has its own simplification needs, and there is considerable variation as to how well the specifics of what makes a text difficult is defined for each group and simplification strategy. While difficult items in a text may be identified more easily and generally for problems such as resolving pronoun reference, questions such as what makes a French word difficult for a native speaker of Japanese, or what dyslexic children consider a difficult character combination or an overly long sentence, are much harder to answer. Nevertheless, there is a vast body of work (Yatskar et al., 2010; Biran et al., 2011; Horn et al., 2014) that ventures to build very general-purpose simplification models from simplification corpora such as the Simple English Wikipedia corpus (Coster and Kauchak, 2011), which has been edited by amateurs without explicit regard to a specific audience, and with rather vague guidelines as to what constitutes difficult or simple language.

Other work in simplification attempts to answer the above questions by inducing models from specifically compiled datasets, which for instance may have been collected by surveying specific target groups and asking them to indicate difficult material in a text. Yet even those approaches often cannot live up to the real challenges in simplification, seeing that we find very heterogeneous simplification needs also within target groups. Foreign language learners with different linguistic backgrounds (pertaining both to their native and second languages) will find very different aspects of the same foreign language difficult. Young readers in different school grades will quickly advance their reading habits and skills, and also within the same class or age reading levels may differ greatly. Likewise, people with autism exhibit very different manifestations of the type and degree of their condition (Alexander et al., 2016), also with respect to reading (Evans et al., 2014), just as there exist many different forms of cognitive impairments affecting literacy, including many different forms of dyslexia (Watson and Goldgar, 1988; Bakker, 1992; Ziegler et al., 2008). In fact, while there is a relatively strong agreement on the existence of some typologies of dyslexia or autism, specific typologies that have been proposed are heavily debated, such that it would not even be straightforward to create simplification tools for specific subtypes of these conditions.

From this it becomes apparent that in order to build simplification systems that truly help specific individuals, those systems have to be personalized or personalizable. Further, due to the frequent lack of insight into what an individual's specific reading problems are (and because any introspection is difficult to verify), such systems need to be able to learn themselves what those individual challenges are, and ultimately adapt to those.

1.2 Obtaining individual data

In order to learn specific reading challenges for an individual person, a simplification system needs individual data for this person, from which a personalized model can then be induced. This brings up the question of how best to obtain such data. A straightforward approach would be to ask each individual to provide ratings for some number of stimuli as they start using a simplification system. However, this would pose a relatively unnatural reading scenario, which might introduce a certain bias in the data and thus distort the induced model. Further, it might create a dissatisfying user experience, and users might not be willing to invest much time into such a calibration phase, especially when they perceive reading as a particularly strenuous activity. Yet perhaps most importantly, the model will not necessarily be well-adapted to the specific domains and genres that a specific user typically consumes text from.

As an alternative, we propose to collect data as the system is used, and to continuously update the system with feedback it collects from the user. In this way, the system can base its model on exactly those text types the user consumes. We discuss how feedback can be incorporated into a system in Section 3 and provide details on how this is implemented in our proposed system in Section 4.

1.3 Contributions

We present Lexi, an open source and easily extensible tool for adaptive, personalized text simplification. Lexi is based on an adaptive framework for lexical simplification that we also describe in this paper. This framework incorporates feedback from users, updating personalized simplification models such as to meet their individual simplification needs. Lexi is made publicly available under a CC-BY-NC license[1] at https://www.readwithlexi.net.

2 Related Work

Perhaps the earliest contribution that focuses on on-demand lexical simplification is the work of Devlin and Unthank (2006), who present HAPPI, a web platform that allows users to request simplified versions of words, as well as other "memory jogging" pieces of information, such as related images.

Another example is the work of Azab et al. (2015), who present a web platform that allows users to select words they do not comprehend, then presents them with synonyms in order to facilitate comprehension. Notice that their approach does not simplify the selected complex words directly, it simply shows semantically equivalent alternatives that could be within the vocabulary known by the user.

The recent work of Paetzold and Specia (2016a) describes Anita, yet another web platform of this kind. It allows users to select complex words and then request a simplified version, related images, synonyms, definitions and translations. Paetzold and Specia (2016a) claim that their approach outputs customized simplifications depending on the user's profile, and evolves as users provide feedback on the output produced. However, they provide no details of the approach they use to do so, nor do they present any results showcasing its effectiveness.

Therefore not counting Paetzold and Specia (2016a) as work in *personalized* simplification, we are not aware of any previous approaches that address this. We further refer to related work on specific aspects of text simplification as they become relevant in the course of this paper.

3 Adaptive text simplification

As we mapped out in the introduction, we devise a simplification system that continuously learns from user feedback and adapts to the user's simplification needs. This section discusses how such feedback can be incorporated into a *lexical simplification* model via online learning, and where in the lexical simplification pipeline it is sensible to implement adaptivity.

3.1 Adaptivity in the lexical simplification pipeline

Lexical simplification, i.e. replacing single words with simpler synonyms, classically employs a pipeline approach illustrated in Figure 1 (Shardlow, 2014a; Paetzold and Specia, 2015). This pipeline consists of a four-step process, the first step of which is to identify simplification targets, i.e. words that the model believes will pose a difficulty for the user. This step is called *Complex Word Identification* (CWI) and has received a great deal of attention in the community, including two shared tasks (Paetzold and Specia, 2016b; Yimam et al., 2018). In a second step, known as *Substitution Generation*, synonyms are retrieved as candidate replacements for the target These are then filtered to match the context, resolving word sense ambiguities or stylistic mismatches, in *Substitution Selection*. Finally, those filtered candidate are ranked in order of simplicity in what is known as *Substitution Ranking* (SR).

Out of these four steps, we consider CWI and SR as the most natural ones to make adaptive, whereas generation and selecting candidates can be regarded as relatively independent from a specific user. In order to implement adaptivity, we propose to make use of online learning methods as discussed below and, departing from a seed model, train and maintain *user-specific models* as we collect feedback.

3.1.1 Adaptive CWI

Complex Word Identification is usually approached as a binary classification task, where the goal is to decide for some word *in context* whether or not it poses a difficulty to a reader. Existing datasets, for instance the ones used at previous CWI shared tasks (Paetzold and Specia, 2016b; Yimam et al., 2018), therefore provide a sentence and a target word (or multi-word expression) together with a binary label.

A model trained on this data with a learning algorithm based on gradient descent on t examples can now easily integrate newly collected data points into its parameters θ using an update rule such as

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta^{(t)}} J(\theta^{(t)}; x, y), \quad (1)$$

[1] https://creativecommons.org/licenses/by-nc/4.0/

where x is a representation of a target word in context and y is a binary complexity label we receive from user feedback. As an alternative to gradient descent based algorithms, we can use other online learning models, e.g. the Perceptron algorithm. CWI datasets are typically not very large (between 2,500 and 5,500 positive examples per dataset in the mentioned shared tasks), such that data points sampled from users can quickly have an impact on a generic base model.[2]

3.1.2 Adaptive Substitution Ranking

Substitution Ranking has received relatively little attention in the community compared to CWI. Most lexical simplifiers rank candidates using unsupervised approaches. The earliest example is the approach of Carroll et al. (1998), who rank candidates according to their Kucera-Francis coefficients, which are calculated based on frequencies extracted from the Brown corpus (Rudell, 1993). Other unsupervised approaches, such as those of Ligozat et al. (2012) and Glavaš and Štajner (2015), go a step further and use metrics that incorporate multiple aspects of word complexity, including context-aware features such as n-gram frequencies and language model probabilities. But even though unsupervised rankers perform well in the task, they are incapable of learning from data, which makes them unsuitable for adaptive SR.

Our approach to adaptive SR is similar to our approach to adaptive CWI, namely to train an initial model over manually produced simplicity rankings, then continuously update them with new knowledge as Lexi users provide feedback on the simplifications they receive. The feedback in this scenario is composed of a complex word in context, a simplification produced by Lexi, and a binary rank provided by the user determining which word (complex or simplification) makes the sentence easier to understand. For that purpose, we need a supervised model that (i) supports online learning so that it can be efficiently updated after each session, and (ii) can learn from binary ranks.

Paetzold and Specia (2017) offer some intuition on how this can be done. They exploit the fact that one can decompose a sequence of elements $\{e_1, e_2, ..., e_n\}$ with ranks $\{r_1, r_2, ..., r_n\}$ into a matrix $m \in \mathbb{R}^{n \times n}$, such that $m(i, j) = f(r_i, r_j)$, and function $f(r_i, r_j)$ estimates a value that describes the relationship between the ranks of elements e_i and e_j. For example, f could be described as:

$$f(r_i, r_j) \begin{cases} 1 \text{ if } r_i < r_j \\ -1 \text{ if } r_i > r_j \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

The ranker of Paetzold and Specia (2017) uses a deep multi-layer perceptron that predicts each value of m individually. It takes as input feature representations of e_i and e_j, and produces a function f similar to the one depicted in Equation 2. Their approach would be perfectly capable of learning from the feedback produced by Lexi users, but it would be very difficult to train it through online learning, given that deep multi-layer perceptrons are characterized by a large number of parameters that are costly to optimize in an on-demand basis. We instead propose to employ an online learning model that has fewer parameters, e.g. logistic regression.

4 Implementation

Lexi consists of a client-side frontend and a server-side backend that communicate with each other via a RESTful API (Fielding, 2000), exchanging requests and responses as described further in 4.3. The client-server architecture allows for easy portability of the software to users, minimizing user-side installation efforts, hardware usage and dependencies on other libraries. It also centralizes the simplification engine, such that amendments to and maintenance of the latter need only be implemented on the server side.

Lexi is currently limited to performing lexical simplification. Note, however, that this is merely a limitation of the backend system, which only implements a lexical simplification system for now. From the frontend perspective, however, there are no limitations as to the nature and length of the simplified items in a text, and extending Lexi to support higher-level modes of simplification simply amounts to

[2] An alternative to traditional, one-size-fits-all approaches has recently been proposed by Bingel et al. (2018), who use eye-tracking measures to induce personalized models to predict misreadings in children with reading difficulties.

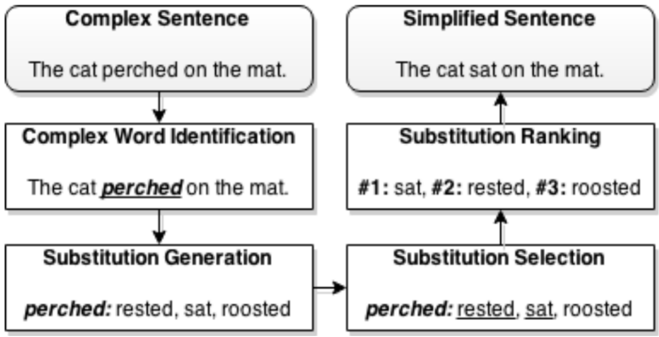

Figure 1: Lexical simplification pipeline as identified by Paetzold and Specia (2015). The simplification workflow consists of identifying simplification targets, i.e. words that pose a challenge to the reader. In the generation step, possible alternatives for each target are retrieved, which are then filtered in the selection step, eliminating words that do not fit the context. In the ranking step, the system finally orders the candidates by simplicity.

implementing a backend system supporting this.[3]

We initially focus on lexical simplification for a number of reasons: (i) We have existing baseline models that we expect to work well in a real-world setting. (ii) Given a relatively small number of parameters in those models, we expect fast adaptation to individual users from relatively little feedback. (iii) Compared to other forms of simplification, lexical simplification needs to make a selection from a relatively limited search space that is still reasonably diverse, such that we expect personalized models to make a difference more easily.

4.1 Frontend

Lexi's frontend is implemented in JavaScript and jQuery under the Mozilla WebExtension framework, supported by most modern browsers.[4] WebExtensions employ *content scripts* to modify a webpage upon certain specified events, for instance a click on some page element. The remainder of this section describes Lexi's basic usage as the user registers an account and asks the system for simplifications, thereby illustrating the user interface and sketching the inner workings of the frontend.

4.1.1 User log-in and registration

Upon installation of the Lexi extension in the browser, the user is prompted to register an account, providing an email address as well as basic demographic information (year of birth and educational level, see Figure 2). This information is sent to the backend using its registration endpoint (see Table 1). If the user has previously created an account and simply reinstalled the extension, they may also just provide their email address to keep using their existing profile. The user's email address is stored locally in the browser, where it is kept until the browser storage is cleared or the extension is uninstalled.

4.1.2 Simplification requests and display

Whenever the user visits a webpage, the extension injects an event listener into the page, which triggers upon the selection of some text and offers the user to simplify the selected content in the form of a small button that is displayed just above the selection. When this button is clicked, the extension retrieves the user's email address from the browser storage (prompting the user to log in if no email address is stored) and verifies that a user with that email address exists in the backend's database, using the login endpoint as given in Table 1. The script then submits a *simplification request* to the backend's simplification endpoint, enclosing a JSON object that contains the user's email address (used by the backend to retrieve

[3]Note that, in general, this paper describes the Lexi frontend and backend versions 1.0. Both parts of Lexi are under ongoing development, with details pertaining to the implementation possibly subject to change.

[4]`https://developer.mozilla.org/en-US/Add-ons/WebExtensions`

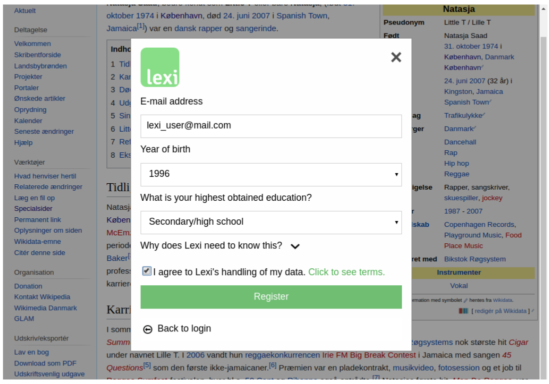

Figure 2: User account registration form

Figure 3: Five-point rating form

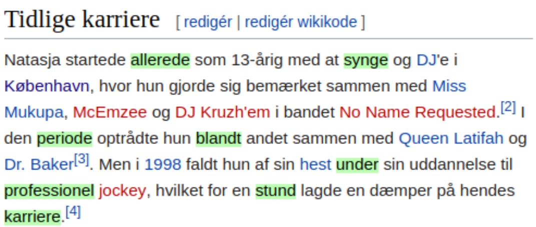

Figure 4: Simplification spans are marked up in light green. As the user clicks on a simplification span, the currently displayed word is replaced with an alternative.

the personal simplification model) and the HTML code of the element containing the text selection. See Appendix A.1 for an example.

The response from the backend then transmits a JSON object with augmented HTML, where <span> elements with unique IDs are wrapped around simplification targets. The response object further contains an array of *simplification objects*, each of which in turn contains a list of synonyms ordered by simplicity ranking (including the target). An example is given in Appendix A.2. The content script replaces the original source with the augmented HTML and displays each simplification span with a light green background color (see Figure 4). The script then shifts through the simplification alternatives for a given target whenever the user clicks on the respective span on the page, advancing one alternative per click and reverting to the first alternative at the end of list. The original item is marked in a slightly but discernibly darker shade than the proposed simplifications.

4.1.3 User feedback

In order to provide personalized simplifications and to adapt to individual users, Lexi needs to be able to decide which alternative a user prefers over the others for every target. In a classical, controlled annotation setting, one would probably present subjects with a set of alternatives and have them rank these or pick a single favorite. However, as Lexi aims to provide as natural and smooth a reading scenario as possible to its users, explicitly asking for such feedback would critically obstruct the reading process.

Lexi therefore interprets whatever final selection a user makes for some simplification span as their preferred alternative in this context.[5] As the user finally navigates away from the webpage that Lexi was invoked on, Lexi solicits feedback from the user on a five-point scale (see Figure 3) and submits this rating along with the simplification objects and their final selections (and click-through counts) to the

[5]In the instructions, users are made aware of this. The frontend further keeps track of how many times the user clicked on a given simplification span, thus providing the backend with information such as how many times the user clicked through the entire list, or whether perhaps no alternatives were solicited for some item.

feedback endpoint of the backend.[6] See Appendix A.3 for an example of the feedback.

4.1.4 Qualitative evaluation of usability

The frontend design was developed in close collaboration with Nota, the Danish Library and Expertise Center for people with reading disabilities.[7] In February 2018, the software was intensively tested by four dyslexic members of Nota, all female students in secondary/higher education and aged between 20 and 30. Each test started with a short preliminary interview in which the subjects were asked about their age, occupation/study field, reading habits, degree of dyslexia and use of browser extensions. The subjects were then given the possibility to watch an introduction video (of 1:30 min length) outlining Lexi's basic functionality and user interface. Two of the four subjects opted for this, while the other two decided to skip the video as they do not usually watch introduction videos when using new software. Next, the subjects were asked to locate Lexi in the Chrome Webshop, install it in the browser and create a user account. Once set up, each subject navigated to a site of her choice and used Lexi to receive simplifications as outlined in 4.1.2. The two subjects who had not watched the video did so now, and both declared they gained further insight into Lexi's functionality through the video, but that it was not crucial in order to understand its basic usage.

In qualitative interviews directly succeeding each test, the test subjects overall reacted very positively to the prospect of a personalized simplification tool in general, and to Lexi and its design in particular.[8] The test subjects suggested a number of improvements, most of which have now been implemented. One suggested improvement, which we have not been able to implement but intend to do so for a future version, is the support for multilingual simplification. Two subjects said they would greatly appreciate this, as much of their study material is only available in English.

4.2 Backend

Lexi's backend consists of a simplification system, implemented in Python 3.5, and a database that stores user information and their simplification histories.

4.2.1 Simplification system

As stated above, Lexi's simplification system currently focuses on lexical simplification, abiding to the *de-facto* standard pipeline depicted in Figure 1. Since Lexi lets users choose which words they wish to have simplified, it does not employ any automatic CWI.[9] Below we sketch Lexi's simplification system as it receives simplification requests from the frontend. As our lexical simplification approach is sensitive to the context of a word, Lexi's first step is to preprocess the HTML source transmitted from the frontend, identifying the boundaries of the sentence that contains the target word, if any.[10]

For Substitution Generation, Lexi implements the embeddings-based approach inspired by the contributions of Glavaš and Štajner (2015) and Paetzold and Specia (2016c). In their work, they extract as candidate substitutions the N words with the highest cosine similarity with a target word. As Danish, the language currently served by Lexi, is not as well-resourced as for example English, Lexi extends the embedding-based Substitution Generation approach by using an ensemble of embeddings models that are trained independently on different text sources, the Danish Wikipedia and a news corpus.[11] The overall similarity score for a target-candidate pair is then defined as the mean score across these embeddings models. Lexi returns the ten most similar candidates whose mean similarity score exceeds some

[6] More correctly, feedback is not solicited when the user actual navigates away from the page, as security restrictions in browsers disallow custom scripts to run upon closing a page. Instead, Lexi asks for feedback via a small notification box in the upper right corner of the page, which pops up as the operating system's *focus* changes to a different window, or when the mouse leaves the browser's viewport (e.g. for the address bar).

[7] `http://www.nota.dk`

[8] An informal evaluation of the software on a 5-point scale (with 1 being worst and 5 best) yielded two ratings of 5, one 4 and one 3.

[9] We do plan, however, to implement CWI as the user solicits simplifications for longer text passages or entire pages.

[10] In order to reduce bandwidth and modify the page more easily, the frontend only transmits the HTML source of the least HTML node fully containing the selection, which typically is a paragraph (`<p>`), but may also be a single word contained in a heading (e.g. `<h1>`), in which case no context is available. Sentence boundaries are identified using NLTK.

[11] `https://ordnet.dk/korpusdk`

configurable threshold. Alternatively, Lexi allows to generate synonyms from a simple dictionary, in the case of Danish using the Danish WordNet (Pedersen et al., 2009), yet this approach suffers from severely reduced coverage compared to word embeddings.

Once generated, the candidates are filtered during Substitution Selection by an unsupervised boundary ranker (Paetzold and Specia, 2016c). In this approach, a supervised ranker is trained with instances gathered in an unsupervised fashion: we generate candidate substitutions for complex words using our generation approach, then assign label 1 to the complex words and 0 to the generated candidates. The boundary between the two classes is then used to rank and filter candidates. Paetzold and Specia (2016c) show that this is a state-of-the-art approach that outperforms all earlier supervised and unsupervised strategies. Given a target word and a set of generated candidate substitutions, the model ranks the candidates based on how far in the positive side of the data they are, then selects 65% of the highest ranking ones.

Finally, the selected candidates are ranked with a supervised Substitution Ranking model following the approach we outlined in Section 3.1.2. It is during this step that Lexi is capable of producing customized output based on the user's needs, and to evolve based on the user's feedback. Lexi employs a pairwise online logistic regression model that learns to quantify the simplicity difference between two candidate substitutions. Given an unseen set of candidate substitutions, the regressor estimates the simplicity difference between each candidate pair, then ranks all candidates based on their average score.

Note that the user's feedback, sent by the frontend, consists of a set S and an index i, where S is the full set of suggested synonyms, including the target, and i is the index of the item in S that the user finally selected. As the regressor, however, learns from pairwise rankings, Lexi passes all pairs $\{\langle S_i, S_j\rangle | j \neq i\}$ to the regressor, i.e. it pairs the selected item with all others and updates the ranker accordingly, postulating that the selected item is easier for this user than each other suggestion.

Using a seed dataset of complex-simple word correspondences in context, we train a default model that produces initial simplifications as a user solicits simplifications for the first time.[12] As Lexi receives feedback for this user for the first time, the seed model is copied and personalized with the first batch of feedback, then this model is saved for later requests by this user.

4.2.2 Database

Lexi stores user information and simplification histories in a PostgreSQL database. More specifically, it employs three different tables, called `users`, `models` and `sessions`. In the first of these, it links a unique, numerical user ID to a user email address, and stores when the user first and last used Lexi. It further contains the demographic information the user provides at registration, i.e. their year of birth and educational status. The `models` table stores a path to the serialized personal model for each user ID. Finally, the `sessions` table stores each simplification request issued to the backend with a unique session ID, the respective user ID, a time stamp for the session start and one for the submission of feedback, the webpage URL, simplification objects serialized as JSON, the provided rating and finally the frontend version number used in this session.

4.3 Communication between backend and frontend

Lexi's backend offers a RESTful API implemented in Python 3.5, using the Flask package.[13] The services available through HTTP POST requests, with their URI paths listed in Table 1. Input and output values are communicated via a JSON-based protocol exemplified in the appendix. Lexi further defines a set of error codes for easier troubleshooting and flexible internationalization of the frontend via the i18n API used by WebExtensions.

[12]Such a seed dataset is not necessarily available for any language. However, in its absence, a seed model could either be trained with simple heuristics, e.g. replacing infrequent words with higher-frequency synonyms. Alternatively, the system could choose to initially rank candidates with such a heuristic and only start learning once the first feedback is available.

[13]`http://flask.pocoo.org/`

URI path	Input	Returns
`/simplify`	User ID; page HTML	Augmented HTML, simplification objects
`/login`	User email address	If successful: User ID, else error code
`/register`	User information	If successful: User ID, else error code
`/feedback`	User ID; simplification objects updated with selections; rating	Status code (successful update or error)

Table 1: RESTful API endpoints defined by Lexi's backend.

4.4 Language support and extensibility

Lexi's design does not impose any restrictions on the support of new (written) languages, including right-to-left or non-alphabetic writing systems. In fact, supporting a new language simply amounts to providing a new language-specific simplification pipeline as illustrated in Figure 1.

Depending on the specific implementation of the simplification system, certain resources are however needed to induce a first seed model for simplification. Most centrally, this pertains to Substitution Generation, where a synonym database or good word embeddings are required in the case of lexical simplification, or a reliable paraphrase module in the case of higher-level simplification. With respect to Substitution Ranking, the availability of resources such as simplification corpora is less critical, as simple heuristics (e.g. simplicity proxies such as length and frequency) might give a reasonable baseline upon which the system can then improve through user feedback.

Lexi currently does not offer multilingual support, but is confined to one language per backend instance. Supporting multilingual simplification could be implemented through a language identification module upstream to the set of simplification pipelines, consisting of one pipeline per language. This raises the interesting question whether knowledge about one user's simplification preferences in one language could be transferred to another language. Support for this hypothesis comes, among others, from the cross-lingual track in the recent CWI shared task by Yimam et al. (2018).

4.5 Ethical and legal considerations

As any software interacting with users and storing information on them, Lexi is naturally subject to ethical and legal concerns, especially those regarding privacy. The EU General Data Protection Regulation (GDPR), for instance, defines a number of regulations such as the clear statement of terms and conditions or that users need to be provided, upon request, with full access to whatever data is stored on them. Lexi does not explicitly store users' names, but in many cases they will be encoded in email addresses. Personally identifiable information may also be stored in the form of simplified text that is logged in the database, for instance if Lexi is used on a user's personal social media profile. The above also highlights the need for encrypted communication between the client and the server, which is safeguarded through TLS encryption over the HTTPS protocol.

Ethical concerns pertaining to text simplification arise when infelicitous simplifications distort the meaning of a text and thus potentially misinform the reader. This is difficult to completely rule out, such that the user should clearly be informed of this possibility. Other concerns revolve around the hypothesis that reducing text complexity will "dumb down" the material and keep users at a low reading level by under-challenging them (Long and Ross, 1993). However, as Rello et al. (2013) point out, "anything which might help [dyslexics] to subjectively perceive reading as being easier, can potentially help them to avoid this vicious circle [of reading less and staying on a low reading level], even if no significant improvement in readability can be demonstrated."

5 Availability and applications

The Lexi software and code, including its backend and frontend, are freely available for non-commercial use under a CC-BY-NC license, obtainable at `https://www.readwithlexi.net`. Researchers can set up their own, customized version of the software and distribute the browser extension to users. It is straightforward to modify features of the software such as offered languages or the exact resources used to induce the initial models.

Besides its core functionality, which we mapped out in the previous sections, Lexi has a number of alternate use cases, which we discuss in this section.

Preloaded simplifications Lexi's primary use case, as described earlier, is to provide simplifications to users as they select a span of text, which circumvents the need for a CWI module as only such items are simplified that the user explicitly solicits replacements for. Alternatively, users may wish to have the entire page simplified before they start reading. Lexi currently implements this functionality, letting the user solicit simplifications for the entire site via a click on the Lexi icon. As there is no personalized CWI module implemented yet, simplification targets are identified via a confidence threshold during Substitution Generation.

Evaluation of simplification quality Via its rating function (Figure 3), Lexi continuously tracks user satisfaction as a means of evaluating synchronic simplification quality as well as the diachronic development of model adaptation. An adaptive model that is continuously customized is expected to gradually improve the average rating it receives from the user.

Data collection Lexi makes it possible to collect user choices over a longer period in order to create bigger simplification datasets. If sufficiently homogeneous subgroups can be identified across users, this data may give insight into their simplification needs, to build better simplification models for them.

Other plausible approaches may understand different users as different *tasks* and apply multi-task learning methods to transfer knowledge between users, thus both regularizing the models for the individual user and increasing the available amount of data that the individual models can be learned from.

6 Conclusion and Future Work

This paper is a first work in personalized, adaptive text simplification, a direction of research motivated by the observation that generic, user-independent simplification systems cannot fully unfold their potential in making text simpler for specific end users. We propose a framework for adaptive lexical simplification, outlining how user feedback can be used to gradually enhance and personalize text simplification. As a concrete first solution to the problem, we present Lexi, an open-source tool for personalized, adaptive text simplification that has been very positively evaluated in a first usability test. In its current implementation, Lexi focuses on lexical simplification in Danish. An extension to other languages is simple, requiring only a medium-sized monolingual corpus on which a language model and word embeddings can be trained.

In future work, we aim to extend the proposed framework to sentence-level simplifications. We further plan to implement support for multilingual simplification.

References

Regi Alexander, Peter E Langdon, Verity Chester, Magali Barnoux, Ignatius Gunaratna, and Sudeep Hoare. 2016. Heterogeneity within autism spectrum disorder in forensic mental health: the introduction of typologies. *Advances in Autism*, 2(4):201–209.

Mahmoud Azab, Chris Hokamp, and Rada Mihalcea. 2015. Using word semantics to assist english as a second language learners. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 116–120, Denver, Colorado, June. Association for Computational Linguistics.

Dirk J Bakker. 1992. Neuropsychological classification and treatment of dyslexia. *Journal of learning disabilities*, 25(2):102–109.

Joachim Bingel and Anders Søgaard. 2016. Text simplification as tree labeling. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 337–343.

Joachim Bingel, Maria Barrett, and Sigrid Klerke. 2018. Predicting misreadings from gaze in children with reading difficulties. In *Proceedings of the Thirteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 24–34.

Or Biran, Samuel Brody, and Noémie Elhadad. 2011. Putting it simply: a context-aware approach to lexical simplification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 496–501. Association for Computational Linguistics.

John Carroll, Guido Minnen, Yvonne Canning, Siobhan Devlin, and John Tait. 1998. Practical simplification of english newspaper text to assist aphasic readers. In *Proceedings of the AAAI-98 Workshop on Integrating Artificial Intelligence and Assistive Technology*, pages 7–10.

Kevyn Collins-Thompson. 2014. Computational assessment of text readability: A survey of current and future research. *ITL-International Journal of Applied Linguistics*, 165(2):97–135.

William Coster and David Kauchak. 2011. Simple english wikipedia: a new text simplification task. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 665–669. Association for Computational Linguistics.

Jan De Belder and Marie-Francine Moens. 2010. Text simplification for children. In *Prroceedings of the SIGIR workshop on accessible search systems*, pages 19–26. ACM.

Siobhan Devlin and Gary Unthank. 2006. Helping aphasic people process online information. In *Proceedings of the 8th SIGACCESS*, pages 225–226.

Richard Evans, Constantin Orasan, and Iustin Dornescu. 2014. An evaluation of syntactic simplification rules for people with autism. In *Proceedings of the 3rd Workshop on Predicting and Improving Text Readability for Target Reader Populations (PITR)*, pages 131–140.

Roy Thomas Fielding. 2000. Rest: architectural styles and the design of network-based software architectures. *Doctoral dissertation, University of California*.

Goran Glavaš and Sanja Štajner. 2015. Simplifying lexical simplification: Do we need simplified corpora? In *Proceedings of the 53rd ACL*, pages 63–68, Beijing, China, July. Association for Computational Linguistics.

Colby Horn, Cathryn Manduca, and David Kauchak. 2014. Learning a lexical simplifier using wikipedia. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 458–463.

Anne-Laure Ligozat, Anne Garcia-Fernandez, Cyril Grouin, and Delphine Bernhard. 2012. Annlor: a naïve notation-system for lexical outputs ranking. In *Proceedings of the 6th International Workshop on Semantic Evaluation*, pages 487–492. Association for Computational Linguistics.

Jun Liu and Yuji Matsumoto. 2016. Simplification of example sentences for learners of japanese functional expressions. In *Proceedings of the 3rd Workshop on Natural Language Processing Techniques for Educational Applications (NLPTEA2016)*, pages 1–5.

Michael H Long and Steven Ross. 1993. Modifications that preserve language and content. *Technical Report (ERIC)*.

Gustavo Paetzold and Lucia Specia. 2015. Lexenstein: A framework for lexical simplification. *Proceedings of ACL-IJCNLP 2015 System Demonstrations*, pages 85–90.

Gustavo Paetzold and Lucia Specia. 2016a. Anita: An intelligent text adaptation tool. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations*, pages 79–83.

Gustavo Paetzold and Lucia Specia. 2016b. Semeval 2016 task 11: Complex word identification. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 560–569.

Gustavo Henrique Paetzold and Lucia Specia. 2016c. Unsupervised lexical simplification for non-native speakers. In *Proceedings of the 13th AAAI*, pages 3761–3767. AAAI Press.

Gustavo Paetzold and Lucia Specia. 2017. Lexical simplification with neural ranking. In *Proceedings of the 15th EACL*, pages 34–40. Association for Computational Linguistics.

Bolette Sandford Pedersen, Sanni Nimb, Jørg Asmussen, Nicolai Hartvig Sørensen, Lars Trap-Jensen, and Henrik Lorentzen. 2009. Dannet: the challenge of compiling a wordnet for danish by reusing a monolingual dictionary. *Language resources and evaluation*, 43(3):269–299.

Luz Rello, Ricardo Baeza-Yates, Stefan Bott, and Horacio Saggion. 2013. Simplify or help?: text simplification strategies for people with dyslexia. In *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility*, page 15. ACM.

Allan Peter Rudell. 1993. Frequency of word usage and perceived word difficulty: Ratings of kucera and francis words. *Behavior Research Methods*, pages 455–463.

Matthew Shardlow. 2014a. Out in the open: Finding and categorising errors in the lexical simplification pipeline. In *LREC*, pages 1583–1590.

Matthew Shardlow. 2014b. A survey of automated text simplification. *International Journal of Advanced Computer Science and Applications*, 4(1):58–70.

Advaith Siddharthan. 2014. A survey of research on text simplification. *ITL-International Journal of Applied Linguistics*, 165(2):259–298.

Adel I Tweissi. 1998. The effects of the amount and type of simplification on foreign language reading comprehension. *Reading in a foreign language*, 11(2):191–204.

Betty U Watson and David E Goldgar. 1988. Evaluation of a typology of reading disability. *Journal of clinical and experimental neuropsychology*, 10(4):432–450.

Mark Yatskar, Bo Pang, Cristian Danescu-Niculescu-Mizil, and Lillian Lee. 2010. For the sake of simplicity: Unsupervised extraction of lexical simplifications from wikipedia. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 365–368. Association for Computational Linguistics.

Seid Muhie Yimam, Chris Biemann, Shervin Malmasi, Gustavo Paetzold, Lucia Specia, Sanja Štajner, Anaïs Tack, and Marcos Zampieri. 2018. A Report on the Complex Word Identification Shared Task 2018. In *Proceedings of the 13th Workshop on Innovative Use of NLP for Building Educational Applications*, New Orleans, United States, June. Association for Computational Linguistics.

Zhemin Zhu, Delphine Bernhard, and Iryna Gurevych. 2010. A monolingual tree-based translation model for sentence simplification. In *Proceedings of the 23rd international conference on computational linguistics*, pages 1353–1361. Association for Computational Linguistics.

Johannes C Ziegler, Caroline Castel, Catherine Pech-Georgel, Florence George, F-Xavier Alario, and Conrad Perry. 2008. Developmental dyslexia and the dual route model of reading: Simulating individual differences and subtypes. *Cognition*, 107(1):151–178.

A Examples of JSON protocol

A.1 Request sent from frontend to backend

```
{
  'frontend_version': '1.0.0',
  'user': 'lexi-user@mail.com',
  'html': '<p>Natasja startede allerede som 13-årig med at synge og DJ\'e ... </p>',
  'startOffset': 17,
  'endOffset': 25,
  'url': 'https://da.wikipedia.org/wiki/Natasja'
}
```

A.2 Reply from backend to frontend

```
{
  'backend_version': '1.0.0',
  'html': '<p>Natasja startede <span id="lexi_254_1" class="lexi-simplify">allerede</span> som 13-årig med at synge og DJ\'e ... </p>',
  'session_id': 254,
  'status': 200,
  'message': 'Simplification successful',
  'simplifications': {
    'lexi_254_1': {
      'choices': ['allerede', 'bare'],
      'selection': 0,
      'sentence': "Natasja startede allerede som 13-årig med at synge og DJ'e i København, hvor hun gjorde sig bemærket sammen med Miss Mukupa, McEmzee og DJ Kruzh'em i bandet No Name Requested.",
      'word_index': 2
    }
  }
}
```

A.3 Feedback sent from frontend to backend

```
{
  'frontend_version': '1.0.0',
  'user': 'lexi-user@mail.com',
  'html': '<p>Natasja startede <span id="lexi_254_1" class="lexi-
     simplify">bare</span> som 13-årig med at synge og DJ\'e ...
     </p>',
  'session_id': 254,
  'simplifications': {
    'lexi_254_1': {
      'choices': ['allerede', 'bare'],
      'selection': 1,
      'sentence': "Natasja startede allerede som 13-årig med at synge
        og DJ'e i København, hvor hun gjorde sig bemærket sammen med
        Miss Mukupa, McEmzee og DJ Kruzh'em i bandet No Name
        Requested.",
      'word_index': 2
    },
  },
  'rating': 4,
  'url': 'https://da.wikipedia.org/wiki/Natasja',
  'session_id': 254
}
}
```