LaTeXMT: Machine Translation for LaTeX Documents

Calvin Hoy and **Samuel Frontull** and **Georg Moser**
University of Innsbruck, Austria
calvin.hoy@screee.ee, {samuel.frontull,georg.moser}@uibk.ac.at

Abstract

While machine translation has taken great strides in recent years, thanks in large part to transformer language models, tools are designed primarily for plain text, and thus not equipped to deal with complex markup documents such as LaTeX. Not even Large Language Models can reliably handle LaTeX source files, as non-standard structures are not captured by any available training data. Previous attempts to create translation engines for LaTeX either work on compiled documents, rely on document pre-processors which may lose critical semantic elements, or cannot distinguish between text and non-text content. In this paper we present LaTeXMT, a software solution for structure-preserving, source-to-source translation of LaTeX documents. All of the source code to LaTeXMT is provided under the LGPL-3.0 open-source licence and a web version is publicly available[1].

1 Introduction

Much progress has been made in the field of Machine Translation (MT), especially Neural Machine Translation (NMT), over the past decade. In particular, transformer-based language models (Vaswani et al., 2017) have proven highly effective.

At its core MT works only on plain text, and cannot handle additional information generally contained within digital documents in addition to their text content—for example, formatting, images and other non-text structures, or programmatic elements. To leave structural elements in place, it is thus necessary to identify the actual text content and the surrounding structural or programmatic components of the document. Utilities for this purpose have long existed for e.g. the Microsoft Office Open XML format[2], but at present no such system is available for the LaTeX (Lamport, 1986) typesetting system commonly used for scientific documents.

Being plain text, LaTeX source files can be fully transformed using MT models. However, they present a challenge for machine translation in that there is no clear separation between text and non-text elements, with *macros* representing both. Advances in transformer models, often also trained on LaTeX sources, enable state-of-the-art MT to largely preserve standard structures. This extends to generative LLMs, which excel in machine translation (Hendy et al., 2023) and are somewhat effective at source code generation (Jiang et al., 2025). However, even sophisticated LLMs are ultimately little more than text generation engines without true understanding of formal or natural languages; their outputs rely on statistical patterns, thereby demanding extensive training data. Thus, state-of-the-art machine translation may break LaTeX structures in—sometimes subtle—ways. Moreover, they are limited by the languages and the amount of text they can process.

To address this challenge, we developed LaTeXMT, a dedicated software solution for LaTeX document translation based on the idea of separating document text from other elements. This enables us to: (i) accurately translate the textual content of LaTeX source files, (ii) preserve the overall document structure, and (iii) retain inline markup associated with the text, all without relying on machine translation services specifically tuned for LaTeX. LaTeXMT enables the integration of custom MT systems, e.g. for languages that are not supported by LLMs. All of the source code[3] to LaTeXMT is provided under the LGPL-3.0 open-source licence and a web version is publicly available at https://latexmt-informatik.uibk.ac.at and is also showcased in a short demo video[4].

[1] https://latexmt-informatik.uibk.ac.at
[2] https://en.wikipedia.org/wiki/Office_Open_XML
[3] https://github.com/latexmt
[4] https://youtu.be/reXOXoZfw3s

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 739–748
November 4-9, 2025

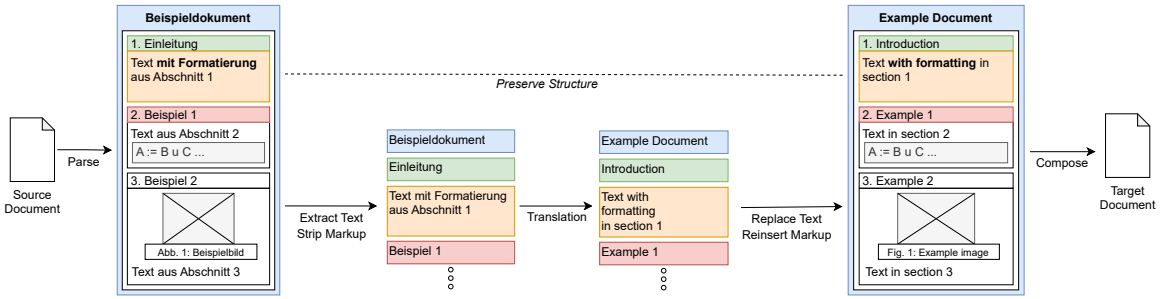

Figure 1: Overview of translation process

Translation Process We have implemented the following pipeline, illustrated in Figure 1:

1. **Parse**: We use the `pylatexenc`[5] Python library to split the source document into a *nested node list*, with each node representing a LaTeX structure such as plain text, macros, environments, or comments.

2. **Extract Text**: We recursively iterate over the *nested node list* to identify contiguous bodies of text (which may consist of multiple paragraphs). We refer to these as *text items*. A *text item* may contain (i) plain text, (ii) *text macros* (e.g. em-dashes, quotation marks, accented characters), (iii) *markup* nodes for formatting (e.g. `\emph`), and (iv) *non-text* nodes. As we want to preserve *non-text* nodes (e.g. inline math) verbatim, we substitute them with *placeholders*, so that they do not interfere with translation.

3. **Strip Markup**: As *markup* nodes within *text items* may be nested, we create a flattened representation of the plain text with markup ranges, which we call a *Markup-annotated String*.

4. **Translation**: The plain text part of each *Markup-annotated String* is translated from the source to the target language. `LaTeXMT` is designed to be agnostic of the translation backend used and supports both local and external services. The local model is provided by the `transformers`[6] library. Moreover, we added support for external APIs, including DeepL and OpenAI, for translation via their respective APIs. Additional translation services can be "dropped in" with minimal or no changes to the core codebase.

5. **Reinsert Markup**: For each text item, a new *Markup-annotated String* is created from the translated text by mapping the position of *markup* nodes from the original text into the translated text using *word alignments*. From there, we construct a new list of LaTeX nodes by linearly scanning over the *Markup-annotated String* and back-substituting *non-text* nodes every time we encounter their corresponding placeholder.

6. **Replace Text**: Finally, each text item's referenced nodes are removed from the original *node list*, and the newly constructed nodes are inserted at the same position. From the modified nodelist, the output LaTeX document is assembled.

The following sections will describe the most important parts of this pipeline in more detail.

2 Text Extraction

We want to create an output document that retains as the original structure. To achieve this, we need to take into account that document text can appear at different nesting levels of a LaTeX document—most commonly at the top level, but also within macros or environments. Structures containing text can also be nested and contiguous text bodies may be interrupted by non-text elements. Thus, we construct an internal representation that preserves the location of document elements while allowing in-place modification of the text. For this, we rely on `pylatexenc`, more specifically its `latexwalker` component. The library provides functionality for parsing a LaTeX document into a nested *node list*, in which environments and macro arguments each hold sub-nodelists representing their contents.

[5] https://github.com/phfaist/pylatexenc
[6] https://github.com/huggingface/transformers

Input	`Write a program $P$ for a register machine $R=((x_i)_{0 < i < 6}, P)$.`
Masking	`Write a program #1_ for a register machine #2_.`
Translation	`Schreiben Sie ein Programm #1_ für eine Registermaschine #2_.`
Re-populate	`Schreiben Sie ein Programm $P$ für eine Registermaschine $R=((x_i)_{0 < i < 6}, P)$.`

Table 1: Preserving non-translatable contents via masking.

2.1 Identifying Document Text

The aforementioned node list does not by itself hold information about which LaTeX structures represent text and which ones do not. While top-level plain text universally represents document text, decisions had to be made on how to interpret other specific structures that we encounter.

Environments We treat the text enclosed within environments as text by default, as we found that to be their most common use, in particular with custom environments. Not all environments do represent translatable document text. Therefore, we provide the ability to specify environments that should be treated differently, e.g. code listings, TikZ graphics, or environments introduced by the `amsmath` packages.

Macros While *macros* sometimes enclose document text, we determined that, unlike environments, it is more common for macros to serve other purposes. Therefore, we do not translate the arguments of macros by default, and maintain two lists for exceptions: one for *markup* macros (see Section 4), and another for macros which enclose independent text elements (e.g. `\section`).

Special characters LaTeX existed in a time before Unicode, aka. before many commonly used symbols (such as quotation marks or German umlauts) could be represented in text files without portability issues. LaTeX therefore has a number of macros and "specials" simply for representing these characters, which we previously referred to as *text macros*. While modern LaTeX engines do all support Unicode, these representations must still be handled by text extraction, so that they may be properly passed to a generic translation service. To obtain a plain Unicode representation of the document text, we extended the `pylatexenc`'s LaTeX-to-text component with a list of common replacements, e.g. for German umlauts (`{\"u}` → ü). This list can be extended with additional replacements as needed.

Based on the decisions above, we recursively traverse the root node list and create a list of *text items*, each of which holds a reference to its position within the node list. These text items are then individually translated, and have their new contents inserted in-place at their original position.

2.2 Preserving Non-text Content

A nearly universal pattern in LaTeX source files is the inclusion of non-text content within the text flow of otherwise plain text, most commonly inline math blocks. Non-text elements generally should not be translated, as most machine translation models cannot handle them. One may simply take the plain text fragments of sentences and translate those individually, but it should be obvious that this is hardly ideal—sentence context is lost, which almost universally leads to sub-par strange translation results. For translation, we must thus consider the sentence as a whole, while hiding its non-text elements from the model—we call this *masking*.

Masking Neural MT models typically retain special symbols, provided they are recognised by the tokenizer, in their original form in the translated output. We leverage this to handle non-text elements by masking them with a *mask*, i.e., a uniquely identifiable sequence of characters. We mask each non-text element appearing within a text body using such a sequence, then translate the masked text bodies, and finally re-populate each mask with their original content, as illustrated in Table 1. We empirically determined the sequence of characters ~N_, which has been found to work well for the Opus-MT models. This sequence is unlikely to occur naturally within a document, and is uniquely identifiable by the literal number N. Importantly, it does not include any words (which may end up being translated), and it starts and ends with different characters.[7] Because different models may handle special character sequences differently, the *mask* can be customised in the user interface to match model-specific requirements.

[7] We initially used the sequence _N_, and found that when two of these occur back-to-back (e.g. _1__2_), occasionally one of the two underscores in the middle is lost in translation.

3 Machine Translation

LaTeXMT is designed to be agnostic of the translation backend used; any translation service can be "dropped in" with minimal or no changes to the core codebase for any language pair supported by a translation backend. Three backends were implemented:

Transformers performs on-device translation using a machine translation model running on the local machine via the transformers Python library.[8] Any translation model supported by the library may be used. For multilingual models, a prefix can be specified to set the target language. We primarily used the Opus-MT models (Tiedemann and Thottingal, 2020; Tiedemann et al., 2023). The `Helsinki-NLP/opus-mt-de-en`[9] model for German to English translation is used by default.

Opus-MT models are trained on corpora with guided alignments, which enables them to emit alignments between input and output tokens for translated text via the transformer attention mechanism. We chose to take advantage of this: when used with Opus-MT models, this translator can simultaneously serve as a word aligner, thus reducing redundant computation and improving processing throughput. When word alignments are available, they can further be used to perform a very rudimentary form of glossary enforcement, where words in the target text are replaced after translation. As this feature may be desirable in certain domain-specific contexts, we extended the translator to include it.

DeepL performs translation off-device via DeepL's public free-tier API, which is limited to processing 500,000 characters per month. The API has built-in support for glossaries.

OpenAI similarly performs translation off-device via OpenAI's API. Translations are obtained by sending a text completion request to a specified GPT model, instructing the language model to translate the text along with an optional glossary. This requires providing a valid API key.

4 Markup Reinsertion

While separating text from non-text during the parsing step is simple enough, macros are commonly used within LaTeX source files for purposes of text markup (aka. formatting), where the macro encloses the text to be formatted—e.g. `\emph{text with emphasis}`. This presents a challenge, as we want to (i) translate the contents of such macros, (ii) translate the complete sentence they appear in, and (iii) convey the intended formatting in the translated document.

Handling markup in the same way we handle non-text macros (via *masking*) may result in poor translations, again due to loss of context. One approach that could be taken to solve this is to train a machine translation model which is markup-*aware*, but this would go against our core idea of *not* relying on specialised translation models. We therefore chose to pursue markup *reinsertion* instead, where we: (i) take note of character ranges enclosed by *markup* nodes in the input text (ii) translate the input text without markup (iii) for each range from step (i), find its corresponding output range(s) (iv) re-insert the markups at the identified output positions. We solve (i) with a datastructure we call *Markup-annotated String*, and (iii) with *word alignments*.

Markup-annotated String As we do not work with LaTeX-aware translation backends, we have to strip markup macros from the text prior to translation, leaving only their contents in place. We transform the nested structure of nodes into a flat representation, where information about markups—including the *macro name* and the *span* of characters it encompasses—is stored in a list external to the text body. Both the text body[10] and markup information are encapsulated in a structure we call a *Markup-annotated String*.

Word Alignments In order to accomplish step (iii) of the process outlined above, we make use of *word alignments*. Obtaining these is a common task in NLP, and different methods exist for this purpose. We rely on `awesome-align` (Dou and Neubig, 2021) that splits sentences into words, tokenises them, and then maps tokens to words to obtain word alignments. We adapted the code[11] to treat punctuation as "whitespace" during alignment, as punctuation is rarely marked up with adjacent words unless another marked-up word follows. Once we have determined markup boundaries in the source text and obtained word alignments, we can use these two components to determine which words in the target text correspond to

[8] https://github.com/huggingface/transformers

[9] https://huggingface.co/Helsinki-NLP/opus-mt-de-en

[10] Note that these text items have already been normalised into a Unicode representation with non-text nodes *masked*.

[11] https://github.com/neulab/awesome-align

marked-up words in the source text, and apply the same markup to them.

5 Evaluation

We evaluated LaTeXMT together with four other systems on synthetic test snippets and full LaTeX-documents: a state-of-the-art NMT system (DeepL), a state-of-the-art LLM (GPT-4o), TransLaTeX, another dedicated tool for machine translation of LaTeX[12], and the open-source NMT systems that serve as base models for LaTeXMT. For German to English translation, we used `opus-mt-de-en`[13] (Tiedemann and Thottingal, 2020) and for Italian to Ladin `lld_valbadia-ita-loresmt-N4`[14] (Frontull and Moser, 2024). The source code of the snippets and the references to the full documents and translation outputs are provided in Appendix B.

5.1 Synthetic LaTeX Snippets

We created a small benchmark of short LaTeX snippets to evaluate German to English and Italian to Ladin translation. The snippets incorporate specialised syntax and commands, including markup macros, custom commands, custom environments, mathematical expressions and special symbols.

Results Table 2 presents the obtained results where we distinguish between the following symbols and codes: ✓ means that the LaTeX snippet was translated correctly and compiled successfully, ! that the snippet compiled after translation, but the translation was incorrect or contained errors and ✗ that the snippet did not compile after translation. `N/A` indicates that the system did not support the respective language pair. More specific error codes are used to describe the nature of the errors. ✗ `cmd` and ✗ `env` denote erroneously translated macros and environment names, respectively. ✗ `omit` denotes that critical syntactical elements were missing, ✗ `halu` and ! `halu` that the translation is purely hallucinated, and ! `tex` means that non-LaTeX syntax was emitted. ! `noise` indicates that the translation contained additional, spurious text, ! `rev` denotes the translation requires revision, while ! `markup` denotes incomplete transfer of markup elements. *base* refers to the respective MT models we used for local machine translation with LaTeXMT.

German to English For German to English translation, we observed that DeepL and the Opus model both struggle with translating LaTeX code, particularly when it comes to handling commands (**C**, **G**,**K**) and environments (**B**, **J**) that match the language being translated. Moreover, they occasionally omit code (**H**,**I**), especially when dealing with longer sequences of consecutive macros (**H**), and have difficulty with special symbols. This can lead to inconsistencies in the output that require revision. In contrast, GPT-4o demonstrates good performance in translating LaTeX code and handles most cases effectively. However, the responses from GPT-4o tend to vary significantly, and the system is sensitive to the instructions given in the prompt. Two instances (**J**,**K**) fail to compile after translation due to the inappropriate translation of macros and environments. TransLaTeX and LaTeXMT, on the other hand, prove to be more reliable solutions for translating LaTeX documents. With TransLaTeX, however, the challenge is to find token strings that are reliably retained, because these are used to hide LaTeX-specific code during translation and then restored afterwards. This is also evident in **E**, where this does not work and results in code that cannot be compiled. The other examples that require revision (**A**, **F**, **H**) can be traced back to Google Translate. For LaTeXMT, only one example (**L**) failing to compile after translation, where it erroneously attempted (and failed) to translate the contents of the unknown-to-it `longtable` environment. This is because environment contents are treated as text by default. However, LaTeXMT could easily be extended to handle this environment (and others). While, it successfully preserves structural components, it faces challenges related to the reinsertion of markup elements (**D**,**F**). In particular, nested formatting macros may lead to inconsistencies. Refining the alignment mechanisms to address these issues is left for future work.

Italian to Ladin For Italian to Ladin translation, the base model performs poorly and just generates hallucinated content (**A**–**L**). DeepL and GPT-4o do not support Ladin at all. TransLaTeX allows to define custom translation services. We implemented a custom machine translation service using the same base model as employed by LaTeXMT within the TransLaTeX framework. However, in most cases the output contained non-compilable LaTeX-code

[12] As of 2025, TransLaTeX is the only tool still available among those mentioned in Section 7

[13] https://huggingface.co/Helsinki-NLP/opus-mt-de-en

[14] https://huggingface.co/Helsinki-NLP/opus-mt-it-lld

	German to English					Italian to Ladin				
	DeepL	**GPT-4o**	**TransLaTeX**	**base**	**LaTeXMT**	**DeepL**	**GPT-4o**	**TransLaTeX**	**base**	**LaTeXMT**
A	✓	✓	! rev	✓	✓	N/A	N/A	✗ omit	! halu	✓
B	✗ env	✓	✓	✗ env	✓	N/A	N/A	✗ cmd	✗ halu	✓
C	✓	✓	✓	✗ cmd	✓	N/A	N/A	✗ cmd	! halu	✓
D	✗ cmd	✓	✓	✓	! markup	N/A	N/A	✗ cmd	! halu	! rev
E	! noise	✓	✗ cmd	! rev	✓	N/A	N/A	✗ cmd	! halu	✓
F	! rev	! noise	! rev	! rev	! markup	N/A	N/A	✗ cmd	! halu	! markup
G	✗ cmd	✓	✓	✗ cmd	✓	N/A	N/A	✗ omit	✗ halu	✓
H	✗ omit	✓	! rev	✗ omit	✓	N/A	N/A	✓	! halu	✓
I	✗ omit	✓	✓	✗ arg	✓	N/A	N/A	✗ cmd	! halu	✓
J	✗ env	✗ env	✓	✗ env	✓	N/A	N/A	✗ env	✗ halu	✓
K	✗ cmd	✗ cmd	✓	✗ cmd	✓	N/A	N/A	✗ omit	✗ halu	✓
L	✗ cmd	✓	✓	✗ omit	✗ omit	N/A	N/A	✗ omit	! halu	✗ omit

Table 2: Error codes and descriptions for the evaluation of LaTeX translation.

due to the persistence of special tokens added by TransLaTeX for translation purposes[15]. Among all tested examples, only example **H** successfully compiled. LaTeXMT, on the other hand, produces better translations using the custom MT model for Ladin, and converts most code examples into valid LaTeX, while significantly reducing hallucinations.

5.2 Full-Document Tests

For a test with complete LaTeX documents, we compared the output of LaTeXMT on the conference paper templates for (**T1**) ACL, (**T2**) Springer Nature, (**T3**) ACM, and (**T4**) IEEE against the respective outputs from the latest available release of TransLaTeX and the current free versions[16] of both DeepL and ChatGPT 5 (as of September 2025).

Results DeepL fails all four tests due to its highly restrictive input length limit, delivering only four truncated documents.[17] ChatGPT, similarly, produces incomplete documents in the cases of all documents except **T1**, where a valid and fully translated (including comments) LaTeX file is produced—although earlier runs had yielded semantically incomplete documents, which compile but are missing subsections and figures from the original. TransLaTeX using its default (Google Translate) backend generates a compilable document from **T2** and **T4**, but some of the content near the end of **T4** appears horizontally squashed; **T1**, **T3** both show compilation errors. LaTeXMT produces valid output for all documents except **T3**[18]. **T4** appears correct except for some preamble macros that were seemingly incorrectly transformed and show up before page 1. **T3** fails to compile due to an edge case around custom macros wrapping `\begin{document}` and `\end{document}`.

6 User Interface(s)

We designed LaTeXMT to be extensible and adaptable, both to different backends for translation and word alignment, and to different user interfaces or document sources. Core functionality is thus encapsulated in a Python library. We chose Python because the `transformers` library provides a simple and flexible interface for running machine learning models locally. Libraries for interacting with many public APIs (DeepL and OpenAI in our case) are also readily available. To interface with LaTeXMT, we provide both a command-line interface (CLI), as well as a web interface via the Flask[19] framework.

CLI All translation parameters—location of the input document(s), source and target language, and optionally, output directory[20], a glossary file, and which translation and alignment backend to use as well as parameters for those backends, are specified via command-line arguments.

```
latexmt --src-lang de --tgt-lang en \
  --translator opus \
  --aligner opus \
  --opus-model-base ./opus-mt-de-en \
  --glossary ./glossary.csv \
  input.tex
```

Listing 1: CLI command example

By specifying '-' for the input file name, LaTeX source code will be read from the terminal standard

[15] We have tried three different format strings (`-tf`): default, `={}.{}=`, `~{}.{}_`, `<{}.{}>`, and `+{}-{}+` but none of them brought any improvements.

[16] In this case used via their respective web interfaces.

[17] Used via an API call, there is no such restriction, although the document may be split mid-word at certain boundaries.

[18] With the limitation that comments are not translated.

[19] https://flask.palletsprojects.com/

[20] The default is `./latexmt_out`.

input, and the translated LaTeX source code is emitted via terminal standard output. In this mode, the output directory is only used for supplemental files included via `\input`. Listing 1 shows an example usage for the command-line interface.

Web Interface For the web interface, the translation and alignment backends are configured server-side, and users may select the source and target language as well as a glossary for translation. The web interface provides text translation, and optionally an interface for submitting multi-file document translation jobs. A live version (without document translation) is made available at https://latexmt-informatik.uibk.ac.at.

7 Related Work

Various works have developed methods to address the challenges of translating LaTeX and other structured documents. In the following, we discuss related works, highlighting how they are connected to our approach. Ohri and Schmah (2021) presented a system for translation of mathematical text from English to French. They rely on Pandoc[21] to parse a LaTeX document and later create a new one, which lets them implement translation as a Pandoc filter. Similar to our work, they mask non-text elements contained within passages of text, before passing those passages to the translation service for processing, and back-substitute them afterwards. Their work however does not handle reinsertion of format macros; these are simply removed. *TransLaTeX*[22], seemingly developed originally at the University of Strasbourg in 2023, takes an approach similar to our work. LaTeX documents are parsed using the `TexSoup` library[23], LaTeX structures are remove and the remaining content is then send to an external translation service of choice. TransIns (Steffen and van Genabith, 2021) is a system for translating rich text documents. They rely on the Okapi framework[24] for parsing various document formats (not including LaTeX) for parsing the document into an abstract structure and later reassembling a translated version. Their paper discusses an older strategy, namely *mtrain* for markup reinsertion, which they demonstrate to be highly error-prone. They compare this to their own method, which they call *Complete Mapping Strategy (CMS)*, and which is similar to our *Markup-annotated String* approach in that it uses word alignments to map markup spans from the original to the translated text. Firefox' built-in translation feature[25] is similar to our work—it parses the current page's DOM, performs text extraction, and translates the text while preserving and re-inserting markup based on word alignments. It uses Opus-MT-derived transformer models[26] running on the local machine for translation, and a statistical word aligner, `eflomal`[27].

8 Conclusion

We developed `LaTeXMT`, a software solution which can reliably translate a wide range of LaTeX documents by separating their text content from non-text elements, producing LaTeX source files that faithfully preserve the original document structure. It maintains the relative positioning of content, supports mixed text and non-text elements (*masking*), and restores text formatting (*markup reinsertion*). In order to achieve markup reinsertion, we made use of *word alignments* obtained from the internal state of a neural machine translation model. State-of-the-art NMT models struggle to process LaTeX code effectively. While modern LLMs are better equipped to manage LaTeX, their capabilities are still limited by context size and language support. `LaTeXMT` offers a compelling solution by providing greater controllability and the ability to handle larger documents more efficiently. Through the integration of custom MT systems, users can tailor `LaTeXMT` to their specific needs, extending its applicability across a wider range of languages, also those not supported by LLMs.

Markup reinsertion based on word alignments usually yields good results, but it does not handle e.g. partial word markup, and it of course fails when word alignment does not find a correspondence between source text and its the translation. To address this, further refinement of the alignment mechanism is needed, possibly through the use of static analysis techniques.

Despite these limitations, `LaTeXMT` provides users with the necessary tools to correct errors as they arise, making it the preferred choice for handling complex LaTeX documents.

[21] https://pandoc.org/
[22] https://github.com/Ectalite/translatex
[23] https://github.com/alvinwan/texsoup
[24] https://okapiframework.org/
[25] https://github.com/mozilla/translations
[26] https://github.com/mozilla/firefox-translations-models/
[27] https://github.com/robertostling/eflomal

Acknowledgments

We would like to thank the anonymous reviewers for their work and valuable comments and suggestions, which have greatly helped to improve our presentation and Gernot Baumgartner for his invaluable support behind the scenes.

References

Zi-Yi Dou and Graham Neubig. 2021. Word Alignment by Fine-tuning Embeddings on Parallel Corpora. In *Conference of the European Chapter of the Association for Computational Linguistics (EACL)*.

Samuel Frontull and Georg Moser. 2024. Rule-Based, Neural and LLM Back-Translation: Comparative Insights from a Variant of Ladin. In *Proceedings of the The Seventh Workshop on Technologies for Machine Translation of Low-Resource Languages (LoResMT 2024)*, pages 128–138, Bangkok, Thailand. Association for Computational Linguistics.

Amr Hendy, Mohamed Abdelrehim, Amr Sharaf, Vikas Raunak, Mohamed Gabr, Hitokazu Matsushita, Young Jin Kim, Mohamed Afify, and Hany Hassan Awadalla. 2023. How Good Are GPT Models at Machine Translation? A Comprehensive Evaluation. *Preprint*, arXiv:2302.09210.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2025. A survey on large language models for code generation. *ACM Trans. Softw. Eng. Methodol.* Just Accepted.

Leslie Lamport. 1986. *LATEX: A Document Preparation System*. Addison-Wesley Publishing Company.

Aditya Ohri and Tanya Schmah. 2021. Machine Translation of Mathematical Text. *IEEE Access*, 9:38078–38086.

Jörg Steffen and Josef van Genabith. 2021. TransIns: Document Translation with Markup Reinsertion. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 28–34, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jörg Tiedemann and Santhosh Thottingal. 2020. OPUS-MT – Building open translation services for the World. In *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation*, pages 479–480, Lisboa, Portugal. European Association for Machine Translation.

Jörg Tiedemann, Mikko Aulamo, Daria Bakshandaeva, Michele Boggia, Stig-Arne Grönroos, Tommi Nieminen, Alessandro Raganato Yves Scherrer, Raul Vazquez, and Sami Virpioja. 2023. Democratizing neural machine translation with OPUS-MT. *Language Resources and Evaluation*, 58(2):713–755.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

A Limitations

Over the course of development, we made a number of decisions in approaching issues with no perfect solutions, with trade-offs usually being made in favour of more general solutions as opposed to more specific ones, and to avoid overcomplicating things as a whole. This ultimately left some edge cases we chose not to solve for, some of which we already mentioned, and all of which we will recount below. In this section, we describe these limitations and mention ways in which it might be approached.

Masking of word sequences Sometimes, a non-text element may act as a stand-in for not just a single noun, but several words. For instance, short equations may be used to say "X gleich Y" in German. Using the *masking* approach in this case may lead to an incorrect translation:

```
Prüfen Sie, ob 1$$A = B$$1 ist.
Prüfen Sie, ob #1_ ist.
Check whether #1_ is.
Check whether 1$$A = B$$1 is.
```

In general however, we found *masking* to be an effective strategy for preservation of non-translatable content—and while attempts could be made to solve for this particular edge cases, others most certainly exist, so we chose to accept it as a limitation.

Non-standard macros/environments It is difficult to determine ahead of LaTeX compilation whether or not the arguments to a macro are going to appear as document text. We thus cannot automatically determine which macros should have their contents translated, and rely instead on a list of well-known ones; unknown macros and their arguments are simply left untouched. Perhaps some form of static analysis could be applied, e.g. repeatedly expanding macros per their definition until only well-known macros remain, in order to track how the arguments are ultimately used.

Named Entities Any named entity that appears in the source text and which matches the source language is translated among with the text. This

simply cannot be solved without either making use of a very finely-trained machine translation model, or adjusting the input to demarcate passages of text that should be preserved verbatim (e.g. enclosing it within a non-standard macro).

Partially marked-up words Our alignment-based markup reinsertion method cannot map markup that spans only part of an input word to the corresponding part in the output. However, partial markup reinsertion is theoretically possible using token-level alignments. Still, the granularity of this approach is inherently limited by the model's tokenization, and thus alternative methods would be needed for fine-grained markup handling.

Comments In LaTeX, nothing can actually ever follow a comment within a line. One might thus consider remembering the contents of comments together with the line number they originally appeared on, then simply appending the comment to the same line after translation. However, this idea is complicated by the fact that we do not preserve individual line breaks in their original position, (as these are structurally irrelevant to the LaTeX engine), which shifts line numbers between the input and output documents.

Whitespace The text extraction functionality of `pylatexenc` simply maps all whitespace characters to a plain space, including non-breaking spaces and manual line breaks. While this behaviour could be changed with ease, this leaves the problem that there is no universal way to represent different whitespace characters in a way that will be reliably preserved by machine translation models, as most if not all perform whitespace-delimited word tokenization and do not concern themselves with different kinds of whitespace. Unlike non-text macros, whitespace also never represents any words within the text body. Thus, we cannot use masking to preserve them, as that would most certainly adversely affect translation results. Beyond paragraph splitting, we did not address with this issue, and accepted a lack of whitespace handling as a limitation. Since output documents tend to require manual review anyway, any non-breaking-spaces or manual newlines can be added back in manually after translation.

Code Listings Presently, the LaTeX parser we use represents the contents of code listings simply as a single text node. This could be extended to run another parser, based on the language specified for the code listing (or a "one-size-fits-most" parser if no language is specified), over the contents of code listings, breaking those up into comment and non-comment nodes, then adding the contents of comments to the list of text items to be translated.

B Test Data

This section includes the code snippets and descriptions of the larger test files used for the evaluation discussed in Section 5. The translation outputs of the different models are not included here, but can be found in the `latexmt/test-cases`[28] repository.

Synthetic LaTeX Snippets

A. LaTeX snippet with mathematical expressions

```
Konstruieren Sie eine TM $M$ über
  dem Alphabet $\Sigma = \{\ma, \mb
  \}$, welche die Sprache
\[
  L = \{x \mid x \text{ enthält den
  String $\ma \mb \ma$}\}
\]
akzeptiert.
```

```
Costruisci una Macchina di Turing
  $M$ sull'alfabeto $\Sigma = \{\ma,
   \mb\}$, tale che $M$ accetti la
  lingua
\[
  L = \{x \mid x \text{ contiene la
  secuenza $\ma \mb \ma$}\}
\]
```

B. Custom environment

```
\begin{hinweis}
  Das ist ein \emph{Hinweis} in
  einem \textbf{custom environment}.
\end{hinweis}
```

```
\begin{nota}
  Una \emph{nota} in un \textbf{
  ambiente personalizzato}.
\end{nota}
```

C. Custom command definition

```
\newcommand{\FF}{\mathsf{false}}
\newcommand{\TT}{\mathsf{true}}

Das \textbf{Ergebnis} ist $\TT$.
```

```
\newcommand{\FF}{\mathsf{false}}
\newcommand{\TT}{\mathsf{true}}

Il \textbf{risultato} equivale a $\
  TT$.
```

D. Custom command usage

[28] https://github.com/latexmt/test-cases

```
\hinweis{Das ist ein \textit{Hinweis
  }.}
```

```
\nota{Questo serve da \textit{nota
  }.}
```

E. Special symbols

```
\textit{Kategorisieren} Sie die
  Sprache der Palindrome \"uber dem
  Alphabet $\Sigma = \{\Null,\Eins\}
  $ anhand der \emph{Chomsky-
  Hierarchie}.
```

```
\textit{Categorizza} il linguaggio
  dei palindromi sull'alfabeto $\
  Sigma = \{\zero,\uno\}$
  utilizzando la \emph{Gerarchia di
  Chomsky}.
```

F. Special symbol 2

```
Es kann gezeigt werden dass die
  Sprache \textbf{\emph{nicht} regul
  \"ar} ist.
```

```
Si pu\`o dimostrare che il
  linguaggio \`e \textbf{\emph{non}
  regolare}.
```

G. `align*` with custom macros

```
\begin{align*}
  P & \to \lw \mid \Null \mid \Eins
    \\
  P & \to \Null P \Null \mid \Eins P
    \Eins
\end{align*}
```

```
\begin{align*}
  P & \to \lw \mid \Zero \mid \Uno
  \\
  P & \to \Zero P \Zero \mid \Uno P
  \Uno
\end{align*}
```

H. Sequence of custom macros

```
Testen Sie Ihren Automaten mit den
  Zeichenreihen $\mb\mb\mb$, $\mb\mb
  \ma\mb\mb$ und $\mb\mb\ma\mb\mb\ma
  \mb\mb$.
```

```
Testate il vostro automa a stati
  finiti con le sequenze $\mb\mb\mb$
  , $\mb\mb\ma\mb\mb$ e $\mb\mb\ma\
  mb\mb\ma\mb\mb$.
```

I. Define date

```
\newtheorem*{uebungen}{Übungen}
\date{\today}
```

```
\newtheorem*{esericizi}{Esercizi}
\date{\today}
```

J. Custom environment 2

```
\begin{leer}
  nicht leer
\end{leer}
```

```
\begin{vuoto}
    non vuoto
\end{vuoto}
```

K. Macro for untranslatable content

```
\newcommand{\germanText}{\textbf{
  Dies ist deutscher Text, der nicht
   übersetzt werden kann!}}

Hier ist der deutsche Text:
\germanText
```

```
\newcommand{\italianText}{\textbf{
  Questa frase italiana non deve
  essere tradotta!}}

Ecco la frase in italiano:
\italianText
```

L. `longtable` environment

```
\begin{longtable}{|c|c|c|}
\hline
\textbf{1} & \textbf{2} & \textbf{3}
   \\
\hline
\end{longtable}
```

```
\begin{longtable}{|c|c|c|}
\hline
\textbf{1} & \textbf{2} & \textbf{3}
   \\
\hline
\end{longtable}
```

Full-Documents Tests

R1. ACL Paper Style LaTeX Template `acl_latex.tex` available at https://github.com/acl-org/acl-style-files/

R2. Springer Nature LaTeX Template, available at https://www.overleaf.com/latex/templates/springer-nature-latex-template/myxmhdsbzkyd

R3. ACM Generic Journal Manuscript LaTeX Template, available at https://www.overleaf.com/latex/templates/association-for-computing-machinery-acm-generic-journal-manuscript-template/yffvrvzbhhpt

R4. IEEE Conference LaTeX Template, available at https://www.overleaf.com/latex/templates/ieee-conference-template/grfzhhncsfqn