Lingonberry Giraffe: Lexically-Sound Beam Search for Explainable Translation of Compound Words

Théo Salmenkivi-Friberg[†] and **Iikka Hauhio**[†‡]
[†] Kielikone Oy, Helsinki, Finland
[‡] Department of Computer Science, University of Helsinki, Finland
{theo.salmenkivi-friberg,iikka.hauhio}@kielikone.fi

Abstract

We present a hybrid rule-based and neural method for translating Finnish compound nouns into English. We use a lightweight set of rules to split a Finnish word into its constituent parts and determine the possible translations of those words using a dictionary. We then use an NMT model to rank these alternatives to determine the final output. Since the number of translations that takes into account different spellings, inflections, and word separators can be very large, we use beam search for the ranking when the number of translations is over a threshold. We find that our method is an improvement over using the same NMT model for end-to-end translation in both automatic and human evaluation. We conclude that our method retains the good qualities of rule-based translation such as explainability and controllability while keeping the rules lightweight.

1 Introduction

In this paper, we present a system for translating Finnish compound nouns into English by using a hybrid rule-based and neural method that constructs a trie of possible translation alternatives in a rule-based manner and then selects the translation using beam search.

Unlike in mainstream machine translation research which focuses on sentences and longer texts, our system is instead intended to be used as a fall-back for a dictionary search, providing translations for individual words in case they are not found in the dictionary. This feature is important for languages such as Finnish, Swedish, and German that allow novel *ad hoc* compound words to be formed freely. These compounds are often long and difficult to parse, especially for non-native speakers, rendering traditional dictionary searches unusable for them.

Unfortunately, translating single Finnish compound words into English using current state-of-the-art neural machine translation (NMT) can be quite vexing: at time of writing, the Finnish word ”puolukka-kinuskirahka” (lingonberry caramel quark) gets translated into English as ”lingonberry custard” and ”lingonberry quinusgiraffe” by two popular commercial machine translation systems. These are mistakes no human translator would make, as the Finnish word is unambiguously the compound of ”puolukka” (lingonberry), ”kinuski” (for which caramel is a reasonable translation, even though others can be argued for) and ”rahka” (quark).

“Puolukkakinuskirahka” is a real-world example and a good demonstration of issues neural systems struggle with (cf. Ismayilzada et al., 2024). In principle, translating Finnish compounds to English is an easy task. Both languages share similar rules for simple compounds (see Figure 1) and the constituent parts of the compounds are often common words found in a dictionary or easily translated by an NMT model. However, rule-based translators also struggle with the task, since the compound parts usually have many translations in the target language, and the probability of a wrong translation being chosen grows exponentially as more parts are added to the compound (Forcada et al., 2011; Khanna et al., 2021).

We solve this issue of lexical selection by choosing the best translation given by the rule-based system by scoring the alternatives with an NMT model. Since the number of alternatives can be very large (even hundreds of thousands in some cases), we implement a beam search for searching the space (cf. Cao et al., 2021). We argue that our method combines the good qualities of rule-based translators such as explainability (each word in the translation can be linked back to a dictionary entry) with the versatility of neural methods, not requiring complex rulesets or algorithms for disambiguation.

Section 2 lists prior work with this problem. Section 3 describes our novel methods. Section 4 describes how we evaluated our system on four datasets: a list of food item names, two small forestry-related term banks, and a subset of IATE. Finally, section 5 discusses the next steps and the limitations of our work.

2 Background

In this section, we describe analysis of Finnish and English compound words and cover relevant prior work both in the fields of rule-based and neural machine translation.

Proceedings of Machine Translation Summit XX Volume 1, pages 173–189, June 23-27, 2025

puolukka-kinuskirahka

lingonberry caramel quark

fromage blanc aux airelles et au caramel

Figure 1: An example of compound words in three languages. The structure of the compound is similar for the head-final Finnish and English, while the head-initial French has a reversed structure. Furthermore, French uses gendered and numbered preposition constructions such as “au” and “aux” (a contraction of the plural article and “au”, thus unambiguously referring to the plural “airelles”) and conjunctions such as “et” to convey relationships between components that are left implicit in Finnish compounds. While in this case the use of the dash makes it explicit that “puolukka” and “kinuski” are unrelated and have the same relationship to “rahka”, the added specificity of languages that structure compounds like French cannot be easily deduced from the Finnish or English compound. This makes translating Finnish to English significantly easier than translating it to French.

2.1 Finnish and English Compounds

In this work, we translate expressions that are formed through two mechanisms: compounding or the concatenation of lexemes (producing such constructions as broadsword, single-minded or distance learning) and prefixation or the prepending of a prefix (eg. preschool).

In Finnish, compounds (Finnish: *yhdyssana*) are very common and can be formed productively (Hakulinen et al., 2004, §399). Finnish compound nouns consist of two or more lexemes written together or with a dash, and can be further broken down into so-called “specifier compounds” (*määriteyhdyssana*) consisting of a specifier followed by the head noun, and “sum compounds” (*summayhdyssana*) consisting of two equal parts (Hakulinen et al., 2004, §398). In case of specifier compounds, the specifier can be inflected in any case (Hakulinen et al., 2004, §403).

2.2 Decompounding

As long as the compounds are made of in-vocabulary words, a morphological analyzer such as Voikko (voi, 2025) or Omorfi (Pirinen, 2015) can be used to analyze the compound and return its constituent parts. We call this process “decompounding”. Voikko and Omorfi are both based on a finite-state transducer (FST) (Beesley and Karttunen, 2003) that returns all possible interpretations of the word. The limitation of these tools is that they do not conduct any disambiguation or parsing, i.e., while compound words have a tree-like structure (Hakulinen et al., 2004, §405) (see Figure 2), the tools return a flat list.

The rule-based Finnish–English translator Transmart (Arnola, 1996) always splits compounds in two: the last component and everything else. As Finnish compounds are either symmetric or head-final (Hakulinen et al., 2004, §398), the last part of the compound is often the most important. However, this choice makes translating compounds that have more than two parts impossible unless both parts returned by the analyzer are present in the dictionary.

There are neural alternatives such as Trankit (Nguyen et al., 2021), but presently its Finnish pipeline also returns a flat list while being significantly worse than the FST-based tools.[1] It is also possible to instruct large language models to perform morphological analysis, although the performance on Finnish is still poor (Moisio et al., 2024; Ismayilzada et al., 2024).

2.3 Rule-Based Translation of Compounds

Productively translating Finnish compounds is not a new endeavor, with early attempts such as Transmart (Arnola, 1996) dating back to the 1990s. This was limited by a disambiguation problem that has later been termed lexical selection by Forcada et al. (2011) and Khanna et al. (2021), i.e. selection of the target lexemes corresponding to the source lexemes. While Forcada et al. and Khanna et al. focus on multi-word expressions (MWEs) instead of compound nouns, we argue that their work is still mostly relevant to this work. Khanna et al. present both a data-driven and a rule-based mechanism for lexical selection. The data-driven approach is based on a maximum-entropy model used to generate weighted lexical selection rules. The rule-based approach is a separate dictionary of MWEs that should be translated as a unit, giving the example of ”little brother” and ”big brother” having separate single-word translations in Kyrgyz.

2.4 Constrained Decoding

Constrained decoding is an umbrella term for methods where a generative model such as a large language model is used not to simply generate the most probable text (or in our case, translation) from the set of all possible texts. Instead, at each generation step only a subset of all possible tokens to continue the text with is considered. Constrained decoding has been successfully used for many NLP tasks (Geng et al., 2023). In the field of machine translation, constrained beam search has been used for incorporating glossaries into an NMT system (Hokamp and Liu, 2017; Post and Vilar, 2018; Hu et al., 2019; Hauhio and Friberg, 2024)

In constrained decoding, the constraints are defined using a grammar. If simple enough, this grammar can be represented as a trie (Cao et al., 2021). In more complex situations, a regular expression resulting in a finite-state machine (Hauhio and Friberg, 2024) or a context-free grammar (Geng et al., 2023) can be used. The grammar is used to determine which tokens can be valid continuations for a sequence, and the probabilities

[1]For example, the Trankit Finnish-TDT pipeline analyses “apumekaanikkoaliupseeri” (assistant mechanic NCO) as “apume#kaanikko#aliupseeri” instead of the correct “apu#mekaanikko#ali#upseeri”. Compare with Figure 2.

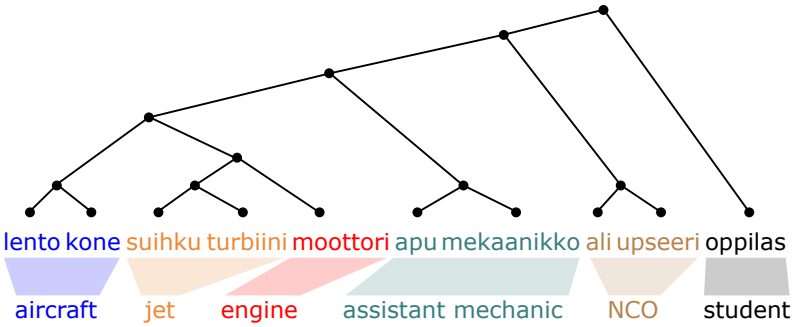

Figure 2: To illustrate the hierarchical nature of Finnish compound words, we use "lentokonesuihkuturbiinimoottoriapumekaanikkoaliupseerioppilas" (aircraft jet engine assistant mechanic NCO student) as an example word. This word is often given in the "longest Finnish words" lists, although it is unclear if it ever has been used in real life (Vartiainen, 2018). While this word is artificially long and would not occur in fluent Finnish text, it is a good example of the recursive nature of Finnish words (Hakulinen et al., 2004, §405). The colored words are found in a dictionary while "konesuihku" (machine jet), "moottoriapu" (motor help), or "mekaanikkoali" (agrammatical) are not. Therefore, merely parsing the word into a flat list of parts is not enough – the tree structure of the word must be maintained such that segments from the dictionary are subtrees of the original tree.

of invalid tokens are set to zero during the decoding. In cases where it is enough for the constraints to appear somewhere in the output such as glossary translation, the grammar contains a wildcard allowing any token to appear. In this work, we use a trie for defining the constraints, and it contains no wildcards, which means that the sequences contained in the trie are the only allowed sequences.

3 Our Method

We present a pipeline for translating Finnish compounds with a hybrid rule-based and neural approach. Our pipeline has the following components:

1. **Morphological Analysis (source language dependent).** This step splits the given word into its component lemmas, returning all possible interpretations of what those lemmas could be. This step results in a list of different ways to split the compound into atomic parts.

2. **Hierarchical Disambiguation (source language dependent).** We decide which atomic compound parts can be combined together into known larger subcompounds that exist in the dictionaries. The result of this step is a list of source components as per Figure 2 (the colored components).

3. **Candidate Generation (target language dependent).** We use a set of rules to generate different spellings, inflections, and other variations of the translations of the compound parts. This accounts for instance for closed versus open compound spelling in the target language.

4. **Token Trie Formation (language independent).** The number of translation alternatives for a compound is at least the product of the number of its components' translation alternatives. This can be in the thousands or tens of thousands, especially when accounting for all the spelling variations we wish to try. To address this, we build a lazily tokenizing trie structure to only construct the translation alternatives that the beam search in the next step will visit.

5. **Beam Search (language independent).** Finally, we search the lazy trie using a beam search to determine a good translation.

3.1 Morphological Analysis

We use the Voikko tool (via the `pyvoikko` Python package) to split the compound words into their constituent parts. As explained in section 2.2, Voikko returns a flat list of the most atomic parts it can find. Knowing the surface forms and order of the parts, we can deduce the corresponding ranges in the string to be analyzed. Some of the ranges may overlap: "maastopaloja" (terrain fires) can be analyzed as *maasto—palo* (terrain fire), *maasto—pala* (terrain piece) or *maa—stop—ala* (earth stop area). For "maastopaloja" we would get the range [1, 3] matching the entry for *maa*, the range [1, 6] matching the entry for *maasto*, the range [4, 7] matching the entry for *stop*, the range [7, 12] twice with one instance matching *pala* and the other *palo* and finally the range [8, 12] matching *ala*.

3.2 Hierarchical Disambiguation

After the analysis, we determine the dictionary translations for the constituent parts. However, in many cases, the dictionary does not only contain the translations for the atomic parts, but also for subcompounds. For example, consider the word "aliupseerioppilas", made of the parts *ali* (sub), *upseeri* (officer), and *oppilas* (student). A word-by-word translation would be "subofficer student". However, the correct translation of *aliupseeri* in English is "non-commissioned officer" or "NCO". Therefore, before determining the translations of the parts, we need

to combine the parts into larger subcompounds in order to find the best translations from the dictionary. This process is further complicated by ambiguity. As "aliupseerioppilas" is made of the subcompound *aliupseeri* and the noun *oppilas*, it should be parsed as *aliupseeri—oppilas*. Interpreting it as *ali—upseerioppilas* is incorrect, even though in our case "upseerioppilas" (officer student) is also present in our dictionary.

Our main solution for the ambiguity is to use the NMT model for choosing the translation from the alternatives using beam search as described in the following sections. However, to reduce the use of resources and ensure useful time-performance, we want the search space to be as small as possible. For this reason, we prune the alternatives using the algorithm described below. We support a case where we have two dictionaries with differing priorities, in our case, a domain-specific glossary and a general dictionary.

1. *Dictionary Query (language independent).* This step looks up translations for the atomic parts and all possible combinations of them in the domain-specific glossary and the general dictionary. For single-letter parts such as "A" in "A-rappu" (A wing), no dictionary query is performed: the single letter itself is returned as the only translation.

   For example, for "aliupseerioppilas", the atomic parts *ali*, *upseeri*, *oppilas*, and the subcompounds *aliupseeri* and *upseerioppilas* are all found in the dictionary. (The full compound *aliupseerioppilas* is not found in the dictionary – if it was, we would return the dictionary translation and not use our system at all.)

2. *Scoring (language dependent).* This step scores translations depending on whether the source term is interpreted as being inflected or not. If it is inflected, it gets a 2.5 point penalty and if its in lemma form, it gets a 1 point penalty. For a given term, only translations with the lowest penalty are kept. In the case of "aliupseerioppilas", all parts are in their base form and receive the penalty of 1.

   We penalize all parts, since we prefer the solutions with fewer compound boundaries and thus longer subcompounds. This is a desirable quality, because compounds can have a different meaning than the sum of their components. Our penalty scheme also prefers doing two extra splits to interpreting a subcompound as inflected, because we find inflected subcompounds to be less likely than a subcompound with a short word that happens to look like a case ending at the end.

3. *Disambiguation (language independent).* This step has two goals: enforcing the domain-specific glossary and finding the best decompounding. We do the first by splitting the compound into dictionary words from the domain-specific glossary and general dictionary so as to maximize the amount of characters covered by words from the domain-specific glossary. We then forget the specifics of what dictionary entry we assigned each range of the compound and consolidate our assigned ranges based on which dictionary they came from. Within each consolidated range, we run another search that finds the decompoundings that have the lowest penalty per step two.

   The word "aliupseerioppilas" does not have domain-specific glossary words, so we treat it as a whole as one consolidated range. We then count the penalties of the possible splits by summing the penalties determined in the previous step: *ali—upseeri—oppilas* has the penalty of 3, while *ali—upseerioppilas* and *aliupseeri—oppilas* both have a penalty of 2. Of these, we return the decompoundings that received the lowest penalty.

   Our analysis would change if we had domain glossary terms. If the user added, for example, "aliupseeri" to the domain-specific glossary, we would have two consolidated ranges: "aliupseeri" and "oppilas". We would then perform this step separately for both of these, but using the glossary for the first range and the general dictionary for the second range. This would result in the single interpretation *aliupseeri—oppilas*. The domain-specific glossary can thus be used for both correcting domain-specific terms and errors caused by incorrect disambiguation.

See Appendix A.1 in general and Algorithms 1 and 2 in particular for a detailed description of this algorithm and Figure 2 for an example of the result of this step.

No edit	"Afro"
Hyphen	"Afro-"
Space	"Afro "
Genitive	"Afro's "
Lower case	"afro"
Lower case + hyphen	"afro-"
Lower case + space	"afro "
Lower case + genitive	"afro's "

Table 1: Different spellings of the word *Afro* generated during the candidate generation.

3.3 Candidate generation

After determining the sequence of compound parts as described above, we generate a list of translation candidates for each part. This is non-trivial, as we have to account for at least capitalization, conveying information from Finnish morphology, and different separators used in different types of English compounds, such as solid, closed, genitive, and hyphenated.

The translation candidates are generated by augmenting the list of dictionary translations determined in the previous phase by including different capitalizations and compound separators: no separator, hyphen ("-"), space

(“ ”), and genitive (“’s ”). For example, the alternatives generated for the word “Afro” are listed in Table 1. The correct spelling and separator depends on the word: e.g., *Afrofuturism*, *Afro haircut*, and *Afro-Asiatic* are all spelled differently. In addition, if the Finnish component was pluralized, we generate the English plural forms of all variations in addition to the singular forms.

The number of alternatives generated can be very high. For example, the Finnish word *afro* has only one translation in our dictionary (“Afro” in different senses), and thus results in the eight alternatives listed in Table 1. However, the word *ali* has five translations (“sub”, “under”, “low-level”, “deputy”, and “hypo”) which result in 40 different alternatives. The word *tulo* has 16 dictionary translations, resulting in 128 alternatives. If the number of alternatives of all the parts in the compound are multiplied with each other, even a word with as little as three compound parts can result in tens of thousands of different possible translations.

To combat this, we prune the number of alternatives by utilizing a list of English prefixes. For words that are not at the end of the compound, we generate candidates as so:

1. We always offer the spelling with a space.
2. We offer the closed compound spelling if the word is on our list of prefixes.
3. We offer the spelling with a hyphen if the word ends with a hyphen.
4. If the Finnish word is plural, we offer the English plural forms. If the Finnish word is in genitive, we offer the English genitive form in addition to the nominative form.

If the word is at the end of the compound, we do not offer the spelling with a space and only offer the spelling with a hyphen if the word itself ends in a hyphen: in that case we assume that the compound ends with some sort of code.

3.4 Decoding

We have two main approaches to decoding. If the number of candidates is smaller than a threshold (less than 400 in our experiments), we rank all of them using the language model in one batch. If we determine that we have too many candidates for this to be efficient, we construct a lazy token trie structure over our candidate translations. Then we implement beam search over the set of all candidates using the trie. Section 3.4.1 describes the token trie structure and section 3.4.2 describes the beam search.

3.4.1 Token Trie

We form a trie of tokens containing all possible tokenizations of the translation alternatives. For example, consider the word “horseshoe”. The optimal SentencePiece tokenization[2] of this word is “_horses hoe”. There is also a multitude of suboptimal tokenizations such as “_horse s hoe”. Our trie includes all possible tokenizations. The rationale for this is that if “horseshoe” is a common word in the model’s training set, the model has likely learned it in the form “_horses hoe”. However, if this was a very rare word, it might be possible that the model has learned its parts “horse” and “shoe”, but not the word as a whole, in which case it makes sense to force a token boundary between the compound parts (“_horse s hoe”). By including the different tokenizations, we allow the NMT model to pick the one it is the most familiar with.

Since the number of combinations is often too large to tokenize at translation time, we perform the tokenization lazily. We form a token trie by first forming a word trie, then a character trie, and finally a token trie. All of these tries are lazy, which means that we only perform tokenization for the branches that are actually searched by the beam search. The details of this step are presented in Appendix A.2.

3.4.2 Beam Search

After forming the trie, we use constrained beam search to search for high-scoring sentences (cf. Cao et al., 2021). In our experiments below, we used a beam size of 50. The beam search is otherwise similar to a regular, unconstrained beam search, but in each step, we remove the hypotheses containing tokens not present in the trie. This limits the output sequences to the sequences encoded in the trie. We also deduplicate the hypotheses such that only the most probable tokenizations per the NMT are retained. The detailed beam search algorithm is presented in Appendix A.3.

4 Evaluation

Our main research question was: “Is this system an improvement over using regular NMT translation for single-word expressions, either in translation quality or in time performance?” To measure the translation quality, we conducted a human evaluation of several compound word datasets translated by both our system and an NMT model decoded with a normal beam search. Along with our human evaluation, we also report automated metrics that we find relevant. Notably, we do not report BLEU as our translation pairs are too short for it to be representative. We also measured the translation time for each of the translated expressions to estimate whether the system is practical.

4.1 Evaluated Systems

We evaluated two systems: our pipeline described in this paper (called “our system” or the “compound translator”), and an NMT model baseline. We ran our pipeline using a commercial dictionary of about 600

[2]Using the English tokenizer of the opus-mt-tc-big-fi-en model.

	Fineli	Hydrology	Forest Soil	IATE	Total
Dictionary hit	118	95	146	1694	2053
No compound translation	19	38	36	114	207
Retained	63	45	20	192	320
Total	200	178	202	2000	2580

Table 2: Terms excluded from evaluation per dataset. Dictionary hit means that the term was found as such in the dictionary. No compound translation means that the term had compound parts that were not found in the dictionary.

000 unique headwords in the Finnish–English direction as the source of compound part translations. This dictionary is designed for human consumption, and no significant changes were made to accommodate the machine translation use case. We scored all alternatives if there were less than 400 and used the beam search otherwise, with beam size set to 50.

We used the Opus-MT Tatoeba Challenge model for Finnish–English[3] (Tiedemann, 2020) as a baseline and also as the scoring model used by our algorithm. For the baseline, we used a beam search of width 50 to match our constrained decoding beam size. We do not compare against other baselines for two reasons: First, the results would not be comparable if the scoring model was different from the baseline model. Second, the chosen model is the state-of-the-art model according to the OPUS-MT Dashboard[4] (Tiedemann and de Gibert, 2023) and choosing a worse model would likely not have given useful information. Furthermore, while we considered using large language models, based on existing literature, they perform poorly in Finnish morphological analysis (Ismayilzada et al., 2024; Moisio et al., 2024) and, while they have demonstrated good translation quality in recent studies (Luukkonen et al., 2024), their inclusion would have required significant computational resources, so ultimately we decided against including them.

4.2 Data

We used the following four test sets for the evaluation:

1. **Fineli Food Composition Database.** We sampled 200 rows from the Basic Package 1 of the national Food Composition Database[5] by the Finnish Institute for Health and Welfare. This dataset includes names of food items. Preprocessing done for this dataset is described in Appendix B.

2. **Forest Hydrology Glossary.** This is a glossary provided by the Finnish Forest Centre. We received an incomplete version of the glossary during its development (Metsäkeskus, 2023a).

3. **Forest Soil Glossary.** As the previous dataset, this was provided by the Finnish Forest Centre in an incomplete state during its development (Metsäkeskus, 2023b).

4. **IATE** We sampled 2,000 terms from the Finnish–English IATE[6]. We used the following parameters when downloading the dataset: all domains, term type term, all reliability levels, and evaluation admitted, preferred, proposed or not specified.

We filtered out terms from the human evaluation in two cases that had to do with the outcome of the dictionary queries. The first case was that in which our system could not parse the term into components found in the dictionary. In real-world use, an NMT translation would be presented to the user. In our experiment, we simply removed these terms.

The much more common case on all data sets was that the compound term was simply found in the dictionary. Our new method reduces to a lemmatizing dictionary search here, both in terms of computational cost and outcome. As such, keeping these terms would be evaluating the contents of the dictionary and not the quality of the system. Again, a real-world user would be presented the lexicographically validated dictionary item that has even stronger accuracy guarantees and upon which a manual intervention is easier. Whereas the first case is quite clearly a undesirable outcome, we see the second case as a not only desirable but even better than coming up with a machine translation at all.

At a deeper level, the rationale for both cases being excluded from our evaluation is that we want to focus on evaluating the compound translator and not the dictionary or the baseline NMT. See Table 2 for the number of terms dropped for the above reasons. Altogether we kept 320 terms.[7]

4.3 Methods

The terms in the datasets were translated with both the proposed system and the Opus-MT model. To measure the improvement in translation quality, we performed both automatic and human evaluation. To measure time performance, we timed the end-to-end time it took for both systems to produce the final translations.

[3] https://huggingface.co/Helsinki-NLP/opus-mt-tc-big-fi-en

[4] https://opus.nlpl.eu/dashboard/index.php?model=top&test=tatoeba-test-v2021-08-07&scoreslang=fin-eng&src=fin&trg=eng (accessed 2025-01-22)

[5] https://fineli.fi/fineli/en/avoin-data (accessed 2025-01-16)

[6] https://iate.europa.eu/home (accessed 2025-01-09); Download IATE, European Union, 2025

[7] Find the evaluated terms at https://github.com/kielikone/mitra-eval-results

4.3.1 Automatic Evaluation

We use the chrF2 (using the `sacrebleu` Python library) and COMET (using the `unbabel-comet` library) metrics. In the case of the Forest Soil data, all our pairs where three words at most, resulting in a BLEU score of 0. As most of our translations are only composed of two or three individual words, we decided against using the word-based BLEU even for datasets where it was non-zero: it would look correct, but not represent the vast majority of the translation pairs.

We used bootstrap resampling to calculate the p-values for the results (using the builtin methods of the `sacrebleu` and `unbabel-comet` libraries).

4.3.2 Human Evaluation

We created an evaluation spreadsheet with rows corresponding to the test words, and columns corresponding to the evaluated systems. We also included a column containing the source language text and another with the reference translation. We randomized the order of the system columns per row. If the NMT model and the proposed system gave the same result, it was presented to the human evaluator only once. If a system gave the reference translation as the result, that translation was not presented to the evaluator and was given the maximal score automatically. If both translations were the same except for differences in capitalization, the one matching in capitalization with the correct translation was presented to the human evaluator and the other was given a grade automatically: if the correctly capitalized got grade 3 (see below), the incorrectly capitalized got grade 2. Otherwise they got the same grade. This deduplication scheme was designed to minimize human labour and systematicise the grading of cases where an obvious grade could be deduced.

We instructed the evaluator to follow the following instructions. The grading scale was devised to allow both comparing the quality of the translations (with the grades having a clear order from worst to best quality) and give useful information regarding the severity of the quality issues relevant for us.

For instance, when translating pikkusuomunokkasärki (a name for the fish species minnow-nase), the following translations would get grades zero to four:

0. *Unsuitable: a translation that has no connection to the original term. 'pinkworm' is an unsuitable translation: it could refer to the name of an animal species, but it has no other commonality with a correct translation.*

1. *Approximate: meaning is conveyed only partially. This category includes translations where the components of the compound have been translated too literally and cases where some part of the compound has been mistranslated. A human that is well-enough versed in the topic can guess what the concept being translated is based on an approximate translation without having knowledge of the source language. 'small scale beak roach' could be an approximate translation: it contains a correct translation for all the components of the Finnish word. A human who knew fish well could perhaps guess at which fish this is: the minnow-nase does seem to have somewhat dense scales, a slightly elongated face and a roach-like body plan generally.*

2. *Spelling mistake: a competent human could produce these words as a translation, but would spell them differently: together / with a space, capitalized / not capitalized, with an accent / without, etc. 'Minnownase' could be an example of a spelling mistake.*

3. *Natural: a competent human could produce this translation. 'minnow-nase' would be a natural translation, as it is the agreed-upon name of this fish.*

Beside the scale, if two system outputs were presented to the evaluator, they were instructed to select one as the better translation if one was better.

Our human evaluator is a linguist with lexicographical expertise employed at our organisation. Evaluation work was carried out during their normal office hours and was compensated according to their normal salary. They are a native Finnish speaker who has an excellent level in English.

We calculated p-values for the results using the Wilcoxon signed rank test.

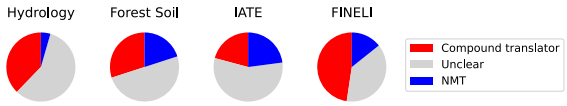

Figure 3: The proportions of terms, for which a system was better than the other according to the human evaluation. This is either inferred from the scores for the translations being different, or directly marked by the evaluator. (See Table 4)

4.4 Results

4.4.1 Automatic Evaluation

The automatic evaluation results are presented in Table 3. With Fineli and Hydrology datasets, the compound translator received significantly higher scores than the NMT baseline, with the chrF2 score increasing from 57.6 to 64.8 (Fineli) and from 47.5 to 52.7 (Hydrology). Similarly, the COMET scores increased from 0.79 to 0.85 and from 0.77 to 0.81, respectively.

On the Forest Soil and IATE datasets, the differences between the two systems were not statistically significant.

4.4.2 Human Evaluation

On the Fineli, Hydrology, and Forest Soil datasets, the compound translator was judged to be better more often

	chrF2			COMET		
	Ours	Opus-MT	p-value	Ours	Opus-MT	p-value
Fineli	**64.8 (64.9 ± 8.2)**	57.6 (57.8 ± 8.6)	0.0060∗	**0.8470**	0.7882	0.0016∗
Hydrology	**52.7 (52.6 ± 9.9)**	47.5 (47.4 ± 9.1)	0.0170∗	**0.8115**	0.7727	0.0090∗
Forest Soil	54.3 (54.6 ± 12.5)	49.9 (50.3 ± 14.9)	0.0949	0.7717	0.7863	0.6104
IATE	50.8 (50.8 ± 4.6)	53.1 (53.2 ± 5.0)	0.0529	0.8012	0.8072	0.3424

Table 3: Automatic evaluation results for the compound translator and the baseline Opus-MT model. For significant results ($p < 0.05$), the better score is bolded.

	Ours	Opus-MT	Unclear
Fineli	**30**	9	24
Hydrology	**17**	2	26
Forest Soil	6	4	10
IATE	40	44	108

Table 4: Number of test words for which the system received a higher grade in human evaluation than the other system. Significant differences are bolded. (See Figure 3)

Dataset	System	Unsuitable	Approximate	Spelling mistake	Natural	p-value
Fineli	Ours	9	15	4	35	0.0000501∗
	Opus-MT	14	13	15	21	
Hydrology	Ours	4	26	0	15	0.00128∗
	Opus-MT	14	18	2	11	
Forest Soil	Ours	3	9	1	7	0.887
	Opus-MT	3	8	2	7	
IATE	Ours	19	107	7	59	0.0584
	Opus-MT	20	100	13	59	

Table 5: The number of each grade the systems received. We used the Wilcoxon signed rank test for significance testing, interpreting our scale as cardinal. We were mainly interesting in whether the difference in grades was statistically significant one way or the other, and so we opted for the two-sided formulation of the test.

	Ours			Opus-MT		
	Avg	Max	Min	Avg	Max	Min
Fineli	0.265	0.715	0.191	**0.157**	**0.417**	**0.111**
Hydrology	0.389	0.622	0.198	**0.138**	**0.402**	**0.0984**
Forest Soil	0.620	0.909	0.410	**0.158**	**0.469**	**0.111**
IATE	0.619	1.06	0.418	**0.137**	**0.393**	**0.0968**

Table 6: Per-term running times (seconds) for compound translation on the different datasets (including the network delays to access the dictionary). Lower time per data set and aggregate function in bold.

than the NMT: 47.6% of terms versus 14.3% (Fineli), 37.8% of terms versus 4.44% (Hydrology); and 30.0% of terms versus 20.0% (Forest Soil). On the larger IATE dataset, the NMT was judged to be better slightly more often than the compound translator: 25.6% of terms versus 20.9%. These results are presented in Figure 3 and Table 4. Like with the automatic evaluation, only the difference in Fineli and Hydrology datasets was statistically significant.

The number of each grade received by the systems is presented in Table 5. We find that the compound translator produced at least as many natural translations as the NMT on every dataset and strictly more on the Fineli and Hydrology terms. On the other end of the scale, it produced no more unsuitable translations than the NMT model on any dataset and again strictly fewer on the Fineli, Hydrology and IATE terms.

We also analyzed how the terms moved from one grade group to another and rendered Sankey diagrams based on this data (see Appendix C). The Hydrology dataset was particularly clean in this, as every translation by the compound translator was rated as at least as good as the corresponding NMT translation (see Figure 8). The case of Fineli (Figure 7), Forest Soil (Figure 9), and IATE (Figure 10) data is more ambiguous, but some patterns do emerge. In particular, terms with spelling mistakes in the NMT translations tend to often flow upward to the highest translation category. In the case of the Hydrology data, spelling mistakes are even completely eliminated. In the IATE dataset, most spelling mistakes from the compound translator were terms that were translated cleanly by the NMT. At least some of these are proper names built up from common nouns (eg. Cohesion Report, Arms Trade Treaty) which have the correct capitalization per the NMT and are spelled fully in lowercase by the compound translator.

4.4.3 Time Performance

The average, maximum, and minimum translation times are presented in Table 6. The compound translator is significantly slower, requiring more than 0.2–0.7 seconds per one test word on average. The Opus-MT NMT model, on the other hand, uses less than 0.2 seconds for each translation on average. There is a significant difference between the Fineli and Hydrology test sets and the Forest Soil and IATE test sets with the latter two requiring over 0.6 seconds on average, while the other two required less than 0.4 seconds on average.

The compound translator makes a request to the dictionary database and performs morphological analysis, both of which take time not required by the NMT model. This added time is an artefact of our test setup and not a constant of our method. Thus, these numbers do not generalize to other setups that, for example, place the database on the same system as the translator and therefore avoid network delay.

5 Discussion

In this section we discuss the implications of our research and take a look at the next steps and further research.

5.1 Scoring is More Practical than Rules

The decoder of a sequence-to-sequence model is a language model and can be used to calculate probabilities of texts. This allows our rule-based component (see Sections 3.2 and 3.3) to be very simple. Instead of focusing on linguistically sound rules that provide good translations, we can just produce as much different alternatives as possible, and filter them with the NMT model. More specifically, we can leave out almost all rules related to disambiguation and focus on rules related to morphological analysis and generation. Since morphological analysis and generation is a much more common task than disambiguation, it is often possible to use preexisting morphological wordlists and libraries (such as Voikko in our case). This radically simplifies the process of writing the ruleset, which is often the bottleneck of development in rule-based machine translation.

Despite our simplified rules, the system maintains, in our opinion, the best qualities of rule-based translators: explainability and controllability. Each translation alternative can be traced back to the dictionary entry that produced it. Furthermore, the user can control the output by editing the dictionary and by deciding which domain-specified glossaries to use.

5.2 The Effect of the Dictionary

The performance of the compound translation is highly dependent on the dictionary. We noticed that the majority of test words in our test datasets were found in the dictionary we used (see Table 2). The compound translator performs better with datasets that have a lower dictionary coverage: the difference between it and the baseline was greatest on the Fineli dataset that had 59% dictionary coverage. Both Forest Soil and IATE had ca. 90% dictionary coverage, but we found no significant difference between the compound translator and the baseline. We argue that while the translator did not improve performance on all datasets, it improved it where it mattered: on the dataset not covered by the dictionary.

While this may be an artifact of the chosen general dictionary and the datasets, we hypothesize that the reason for this might also be that the words in the Forest Soil and IATE datasets are on average more unintuitive and not merely the sum of their parts, which both causes the compound translator to perform poorly, but also is the reason for their inclusion in the dictionary: unintuitive words that cannot be translated productively are more important for the dictionary user. The Fineli dataset, on the other hand, contains names of food items that are typically constructed systematically by listing the same ingredients in all languages.

5.3 Next steps

Currently, our system assumes that the word order of the source and target languages is the same, with a limited number of separators between the compound parts. However, to support languages with differing word order and more complex multi-word expressions (MWE), this assumption does not hold anymore (see Figure 1). This might require replacing the trie structure with a more complex datastructure such as a finite-state machine (FST) (cf. Hauhio and Friberg, 2024) encoding the different word orders.

In this paper, we focused on translating single compound words. However, translating sentences and whole texts is a much more common task in machine translation. We hypothesize that our system could be combined with a terminology-constrained translation algorithm (Hauhio and Friberg, 2024; Bergmanis and Pinnis, 2021; Nieminen, 2024) by scanning the sentence for compounds and adding them as constraints for the translator. In the case of FST-based algorithms, the token trie produced by our system might even be directly incorporated into the finite-state machine.

We hypothesize that many of the issues NMT models have with long compound words are caused by unoptimal tokenization. For example, the word used in the title of this paper, "puolukkakinuskirahka", was translated as "lingonberry giraffe" likely because the model confused the substring "kirah" with the word "kirahvi" *giraffe*. This issue might be mitigated simply by tokenizing the string differently and forcing a token boundary between the compound parts (in this case, between "kinuski" and "rahka").[8] See Section 3.4.1 for more discussion about tokenization.

5.4 Limitations

A major limitation is that we only compared the compound translator to one other system. The choice of the NMT system affects the results considerably. In particular, the model we used has been predominantly trained with full sentences, and its quality may have degraded when applied to single-word source texts. Furthermore, the NMT model was used with a 50-wide beam. It is possible that a greedy search or a smaller beam size could have returned better results (cf. Yang et al., 2018). As noted in Section 5.2, the choice of the dictionary used is also important. In this work, we only used one model and one dictionary, which limits the generalizability of our results.

Another limitation is that our evaluation is relatively narrow: We only used chrF2 and COMET for automatic evaluation, and COMET is intended for scoring full sentences, so it might not be as indicative for single-word translation quality. For manual evaluation, we only had one reviewer. Furthermore, our test datasets were quite small since we had to drop many terms that were already present in our dictionary.

6 Conclusions

In this paper, we presented a system for translating Finnish compound nouns into English using a hybrid rule-based and neural approach. According to our automatic and human evaluation, our system performs better on average than the baseline NMT model on two of our four test sets and has the same performance on the other two test sets. Unlike the NMT model, our system is explainable and controllable, allowing the user to see the dictionary entries the translation is based on, and to fix possible errors by simply modifying the glossary used for translation. While we limited our scope to a single language pair and only compound words, we see promise in our methods to be usable in many kinds of different hybrid rule-based and neural systems.

Carbon Impact Statement

The sum of the translation times of all terms was a hair under 25 minutes, but it does not account for network delays for round trips to the laptop coordinating the translation or partial re-runs that we had to do. In total we estimate that we used no more than an hour of GPU time on an NVIDIA Tesla T4 GPU rated at a 70W maximum power draw to translate a little over 5000 terms. In practice these experiments were running on Amazon g4dn.xlarge instances in the Stockholm region. We used two M3 MacBook Pros and their built-in GPUs and the aforementioned g4dn.xlarge instances for the development of the system and trained no new models. We find it likely that normal development activities on our two laptops, our CI systems and our two g4dn.xlarge staging instances (running mostly idle) over the course of multiple months of development are the larger energetic cost. The model we used is also comparatively small, coming in at 236 megaparameters.

Overall, this results in an approximated less than 60 g CO2e emissions based on the `https://calculator.green-algorithms.org/` online calculator. Of this, most is for the testing during the development of the system and only less than 1 g is for the experiments.

Acknowledgments

Both TS and and IH are funded by Kielikone Oy. IH is additionally funded by the Doctoral Program in Computer Science at the University of Helsinki. We would like to thank Pekka Kauppinen for evaluating the translated compounds, and the peer reviewers for providing valuable feedback and suggestions for further research and possible ablation studies.

[8] We tested this one case and found that "puolukkakinuskirahka" was tokenized as `['_puol', 'ukka', 'kin', 'us', 'kir', 'a', 'hka']`. When we instead tokenized it as `['_puol', 'ukka', 'kin', 'us', 'ki', 'rah', 'ka']`, the NMT model translated it as "lingonberry caramel". While this is not the desired "lingonberry caramel quark", it is better than the translation "lingonberry giraffe" given with the default tokenization. In both of these cases, we used the beam width of 5.

References

2025. Voikko – free linguistic software and data for finnish. https://voikko.puimula.org/. Accessed: 2025-01-13.

Harri Arnola. 1996. Kielikone Finnish-English MT system "TranSmart" in practical use. In *Proceedings of Translating and the Computer 18*.

Kenneth R Beesley and Lauri Karttunen. 2003. Finite-state morphology: Xerox tools and techniques. *CSLI, Stanford*, pages 359–375.

Toms Bergmanis and Mārcis Pinnis. 2021. Facilitating terminology translation with target lemma annotations. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 3105–3111, Online. Association for Computational Linguistics.

Nicola De Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni. 2021. Autoregressive entity retrieval. In *International Conference on Learning Representations*.

Mikel L Forcada, Mireia Ginestí-Rosell, Jacob Nordfalk, Jim O'Regan, Sergio Ortiz-Rojas, Juan Antonio Pérez-Ortiz, Felipe Sánchez-Martínez, Gema Ramírez-Sánchez, and Francis M Tyers. 2011. Apertium: a free/open-source platform for rule-based machine translation. *Machine translation*, 25:127–144.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. 2023. Grammar-constrained decoding for structured NLP tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10932–10952, Singapore. Association for Computational Linguistics.

Auli Hakulinen, Maria Vilkuna, Riitta Korhonen, Vesa Koivisto, Tarja-Riitta Heinonen, and Irja Alho. 2004. *Iso suomen kielioppi*. Suomalaisen Kirjallisuuden Seura.

Iikka Hauhio and Théo Friberg. 2024. Mitra: Improving terminologically constrained translation quality with backtranslations and flag diacritics. In *Proceedings of the 25th Annual Conference of the European Association for Machine Translation (Volume 1)*, pages 100–115, Sheffield, UK. European Association for Machine Translation (EAMT).

Chris Hokamp and Qun Liu. 2017. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1535–1546, Vancouver, Canada. Association for Computational Linguistics.

J. Edward Hu, Huda Khayrallah, Ryan Culkin, Patrick Xia, Tongfei Chen, Matt Post, and Benjamin Van Durme. 2019. Improved lexically constrained decoding for translation and monolingual rewriting. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 839–850.

Mete Ismayilzada, Defne Circi, Jonne Sälevä, Hale Sirin, Abdullatif Köksal, Bhuwan Dhingra, Antoine Bosselut, Lonneke van der Plas, and Duygu Ataman. 2024. Evaluating morphological compositional generalization in large language models. *arXiv preprint arXiv:2410.12656*.

Tanmai Khanna, Jonathan N Washington, Francis M Tyers, Sevilay Bayatlı, Daniel G Swanson, Tommi A Pirinen, Irene Tang, and Hector Alos i Font. 2021. Recent advances in apertium, a free/open-source rule-based machine translation platform for low-resource languages. *Machine Translation*, 35(4):475–502.

Risto Luukkonen, Jonathan Burdge, Elaine Zosa, Aarne Talman, Ville Komulainen, Väinö Hatanpää, Peter Sarlin, and Sampo Pyysalo. 2024. Poro 34b and the blessing of multilinguality. *Preprint*, arXiv:2404.01856.

Metsäkeskus. 2023a. Metsähydrologia-sanasto. Received: 2023-08-04.

Metsäkeskus. 2023b. Metsämaa-sanasto. Received: 2023-11-28.

Anssi Moisio, Mathias Creutz, and Mikko Kurimo. 2024. LLMs' morphological analyses of complex FST-generated Finnish words. In *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics*, pages 242–254, Bangkok, Thailand. Association for Computational Linguistics.

Minh Van Nguyen, Viet Dac Lai, Amir Pouran Ben Veyseh, and Thien Huu Nguyen. 2021. Trankit: A light-weight transformer-based toolkit for multilingual natural language processing. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*.

Tommi Nieminen. 2024. Adding soft terminology constraints to pre-trained generic MT models by means of continued training. In *Proceedings of the First International Workshop on Knowledge-Enhanced Machine Translation*, pages 21–33, Sheffield, United Kingdom. European Association for Machine Translation (EAMT).

Tommi A Pirinen. 2015. Omorfi—free and open source morphological lexical database for finnish. In *Proceedings of the 20th Nordic conference of computational linguistics (NODALIDA 2015)*, pages 313–315.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels. Association for Computational Linguistics.

Matt Post and David Vilar. 2018. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1314–1324.

Jörg Tiedemann. 2020. The Tatoeba Translation Challenge – Realistic data sets for low resource and multilingual MT. In *Proceedings of the Fifth Conference on Machine Translation*, pages 1174–1182, Online. Association for Computational Linguistics.

Jörg Tiedemann and Ona de Gibert. 2023. The OPUS-MT dashboard – a toolkit for a systematic evaluation of open machine translation models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 315–327, Toronto, Canada. Association for Computational Linguistics.

Vilma Vartiainen. 2018. Pisin sana. *Kielikello*.

Yilin Yang, Liang Huang, and Mingbo Ma. 2018. Breaking the beam search curse: A study of (re-)scoring methods and stopping criteria for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3054–3059, Brussels, Belgium. Association for Computational Linguistics.

A Algorithms

A.1 Hierarchical Disambiguation

We call the process that parses our compound into subcomponents found in our dictionaries hierarchical disambiguation, as it respects the hierarchy of the subcompounds (see Figure 2). Besides respecting the hierarchy, we wish the resultant parse to have two other desirable properties: respect our domain glossary and be otherwise linguistically plausible. Our solution to this is composed of two successive dynamic programming algorithms.

The first one or Algorithm 1 concerns itself with enforcing the domain glossary: we consider it less desirable to fail by not using domain glossary terms than parsing the hierarchy suboptimally due to those terms. It is a failure modality that is easily diagnosed by translating without the domain glossary and easily fixed by ammending the domain glossary. Given the ranges in the compound that entries from the domain glossary and entries from the general dictionary represent, Algorithm 1 splits the compound into domain glossary ranges and general dictionary ranges such that the number of characters attributed to domain glossary ranges is maximized. This does not yet tell us what is the best way to parse the subcompounds, only that some solution exists.

The second one or Algorithm 2 is then run individually on each of the subcompounds produced by the previous step. Each run of Algorithm 2 is only given access to terms coming from the dictionary matching that subcompound, thus enforcing the domain glossary whenever some subrange can be parsed with only domain terms. Algorithm 2 produces the parse that minimizes the penalties associated with the ranges.

At the end of this search, we get a sequence of dictionary entries (and the grammatical forms they were inflected in) that match the compound word that was given. Alternatively, we get no sequence at all, indicating that the dictionary we have can not be used to produce a translation for the given compound. This happens early, right after our first dynamic programming search (Algorithm 1). In this situation, we find it wise to fall back onto a normal NMT system, potentially using a constraint-translation method for honoring the domain-specific glossary (cf. Hauhio and Friberg, 2024). If a sequence is produced, it has some desirable qualities. In particular, it represents the interpretation of the compound word that had the most characters covered by domain-specific terms. If there is only one way to choose the ranges to maximally cover from our domain glossary, the parse is further the one considered linguistically the most credible per our scoring. Given this parse, we can proceed onto generating the set of candidate translations, as we see in Section 3.3.

A.2 Token Trie

For the constrained beam search described in Section 3.4.2, we need to know what tokens an incomplete translation can continue with. As we may have exponentially many translation candidates in regard to the number of compound components, we wish to have a data structure that can be built lazily, focusing only on the translation candidates the model seems to prefer. Finally, the data structure needs to somehow map from our compound components to tokens. This is made difficult by SentencePiece, which expects look-ahead to the next piece of punctuation. This can be in the next compound component if we are unlucky. We would need to know the full translation in order to tokenize it. The NMT model is trained on optimal SentencePiece tokenizations, and we do not trust it to rank suboptimal tokenizations reliably.

The solution we present here is a multi-level prefix trie. Using a trie for this kind of constraint decoding is a well-known technique (Hu et al., 2019; Hauhio and Friberg, 2024). Some previous works do incorporate a level of dynamism into the trie itself: Cao et al. (2021) use control tokens to either enable or disable constrained decoding from the trie, effectively amounting to an infinite nested trie structure. However, all previous works we have found either have a small-enough set of options to tokenize them fully at translation time (Hu et al., 2019; Hauhio and Friberg, 2024) or know all the options ahead of time (Cao et al., 2021). Knowing all the items in the trie ahead of time allows Cao et al. (2021) to simply tokenize everything before their inference stage and have tokenization take as much time as it needs. We need to deal with tokenization while we are running

Algorithm 1 The constraint coverage algorithm. Takes as input a sequence of ranges that belong to dictionary entries from a domain-specific glossary, a sequence of ranges that belong to dictionary entries from a general dictionary and the length of the full string these ranges point to. Returns either null if there is no way to select non-overlapping ranges that cover the whole of the string or a selection of ranges that do. If it returns such a selection of ranges, they are each annotated with a boolean value indicating whether it belongs to the domain specific glossary. This selection is guaranteed to have the maximal number of character indices covered by ranges from the domain specific glossary.

```
1:  procedure COVERAGE(glossary_ranges, normal_ranges, length)
2:      coverages ← [0]
3:      solutions ← [([0, 0], False)]                ▷ Here [0, 0] is a range
4:      glossary_lookup ← empty hash map
5:      for all r ← glossary_ranges do
6:          if r.stop ∉ glossary_lookup.keys() then
7:              glossary_lookup[r.stop] ← []
8:          end if
9:          glossary_lookup[r.stop].append(r)
10:     end for
11:     normal_lookup ← empty hash map
12:     for all r ← normal_ranges do
13:         if r.stop ∉ normal_lookup.keys() then
14:             normal_lookup[r.stop] ← []
15:         end if
16:         normal_lookup[r.stop].append(r)
17:     end for
18:     for all i ← [1, text_length] do
19:         best_coverage ← −∞
20:         solution ← null
21:         for all r ← glossary_lookup[i] do
22:                                                  ▷ Note the +|r|
23:             if coverages[r.start] > −∞ ∧ best_coverage ≤ coverages[r.start] + |r| then
24:                 best_coverage ← coverages[r.start] + |r|      ▷ likewise
25:                 solution ← (r, True)
26:             end if
27:         end for
28:         for all r ← normal_lookup[i] do
29:             if coverages[r.start] > −∞ ∧ best_coverage ≤ coverages[r.start] then
30:                 best_coverage ← coverages[r.start]
31:                 solution ← (r, False)
32:             end if
33:         end for
34:         solutions.append(solution)
35:         coverages.append(best_coverage)
36:     end for
37:     if solutions ends with null then
38:         return null                              ▷ No solution was found
39:     end if
40:     selected_ranges ← []
41:     i ← |solutions|-1
42:     while i > 0 do
43:         selected_ranges.append(solutions[i])
44:         i ← solutions[i][0].start
45:     end while
46:     reverse(selected_ranges)
47:                 ▷ Consolidate subsequent ranges with the same source.
48:     return consolidate(selected_ranges)
49: end procedure
```

Algorithm 2 The hierarchical parsing algorithm, made up of two procedures. The main procedure PARSE takes as argument the ranges that are attributed to dictionary entries that are from the correct dictionary and within relevant_range, their matching penalties and the range of the subcompound it is operating in. It returns the set of lists of ranges that the subcompound can be parsed into with the minimal penalty. UNRAVEL backtracks through the dynamic programming data structure and recursively produces the aforementioned set, to be returned by PARSE. It is the caller's responsibility to match the ranges to dictionary entries.

```
1:  procedure PARSE(ranges, penalties, relevant_range)
2:      range_lookup ← empty hash map with default value of []
3:      for all r, p ← zip(ranges, penalties) do
4:          range_lookup[r.stop].append((r, p))
5:      end for
6:      cost_table ← empty hash map with default value of ∞
7:      cost_table[relevant_range.start] ← 0
8:      solutions ← empty hash map with default value of []
9:      for all i ← sorted(range_lookup.keys()) do
10:         for all r, p ← range_lookup[i] do
11:             if cost_table[r.start] + p < cost_table[i] then
12:                 cost_table[i] ← cost_table[r.start] + p
13:                 solutions[i].clear()
14:             end if
15:             if cost_table[i] < ∞ ∧ cost_table[r.start] + p = cost_table[i] then
16:                 solutions[i].append((r, p))
17:             end if
18:         end for
19:     end for
20:     if solutions[relevant_range.stop] = [] then
21:         return ∅
22:     end if
23:     return UNRAVEL(solutions, relevant_range.stop, relevant_range.start)
24: end procedure
25: procedure UNRAVEL(solutions, i, start)
26:     if i = start then
27:         return {[]}
28:     end if
29:     result ← ∅
30:     for rhs ← solutions[i] do
31:         for lhs ← UNRAVEL(solutions, rhs[0].start, start) do
32:             result.add(concatenate(lhs, rhs[0]))
33:         end for
34:     end for
35:     return result
36: end procedure
```

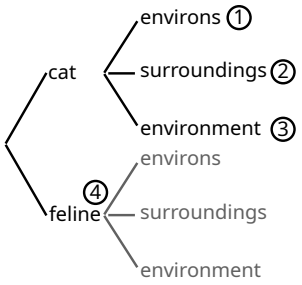

Figure 4: Compound trie. The lazy search has not explored continuations to “feline”. The circled numbers are search states held by the character trie (see Figure 5).

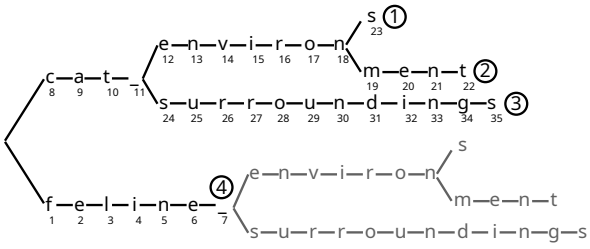

Figure 5: Character trie. The character trie holds references to the compound trie (see Figure 4) in its cells (circled numbers). Note the unexplored continuations of feline.

inference, and so even if our trie is multiple orders of magnitude smaller, it is still a bottleneck.

Our trie has three levels: on the bottom level, we have a lazy search tree in the compound space (see Figure 4). It handles the complexities of generating spelling variations and can generate the set of all follow-up search states with one translation of a compound locked in. It indicates that the search is complete by returning an empty set.

On top of this lazy search tree, we build a character-level trie (see Figure 5). Externally, it has a similar interface: it has a method to list all characters that can continue a string along with indices in the trie that match these continuations. Internally the cells of this trie hold references to the compound search states. If the possible continuations of a cell that holds no references to the compound search are queried, the outgoing edges from that cell are returned. Otherwise, the compound search states are asked to generate their follow-ups and nodes are added to the character trie accordingly. The references that the original cell was holding are cleared and any new cells matching an unexplored search state are pointed to it. The character trie is initialized with a single cell holding a reference to an unexplored compound search tree. Thus we can abstract away the compound formation completely and deal with characters without losing the laziness of the compound search tree.

Finally, on top of the character trie, we build the token trie (see Figure 6). In a similar manner, it exposes a method to list all tokens that can continue a token sequence along with indices into the token trie that match them. Cells in the token trie hold references to the character trie and unravel it just enough to know what tokens can continue a sequence, much like the character trie itself does with the lazy search tree. In practice, the token trie holds a character-level prefix trie of all the tokens in the vocabulary, allowing for fast elimination of tokens that are not allowed given the allowed characters that the character level trie produces. This third level allows us to answer queries about the token-level representation of the translation without fully knowing it in advance: among the options that the token trie generates, the correct SentencePiece tokenization is bound to exist and selecting it is the task the NMT model has been trained on.

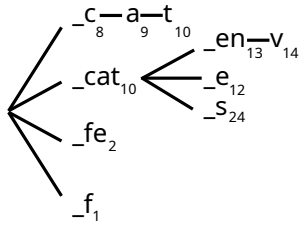

Figure 6: Token trie. The token trie holds references to the character trie (see Figure 5) in its cells (subscript).

A.3 Beam Search

The beam search algorithm is presented as Algorithm 3.

B Data Preprocessing

We sampled 200 rows from the Basic Package 1 of the national Food Composition Database in Finland[9]. It required some non-trivial pre-processing. Each row of the files foodname_FI.csv and foodname_EN.csv in the package contains a description of an ingredient or a food item such as ’FLOUR MIXTURE, FOR BREAD ROLLS, WHEAT FLOUR, WHEAT GROATS, RYE’. We split this string on commas and only kept the first chunk, as it seemed to match better across languages. These string pairs were filtered to only keep the ones where the Finnish side contains only dashes, characters A-Z and Å, Ä and Ö. Selected strings were then lowercased. From the lowercased string pairs, a sample of 200 was randomly drawn. The matching of the Finnish and English terms was somewhat imperfect, as noted by our evaluator.

C Sankey Diagrams

We produced the Sankey diagrams in Figures 7, 8, 9, and 10 to visualize how our system changed the translations when compared to the NMT model. The diagrams have the evaluation result of the NMT model on the left and our system on the right.

[9] https://fineli.fi/fineli/en/avoin-data

Algorithm 3 Constraint beam search algorithm used in this work. Unlike the ones used in (Hauhio and Friberg, 2024), (Hu et al., 2019) and (Post, 2018), it is rooted such that the constraint starts at the start of the string and spans the whole string. Takes as arguments the token trie object, the width of the beam and a function LM that calls the language model on token sequences. Returns a tuple with the tokens of the best hypothesis as the first element and its score as the second. The book-keeping to match the selected dictionary entries to the hypotheses is ommitted for brevity. In practice, the token trie would know the matching hierarchical analysis for any final state. The analysis for a finished translation can then be deduced by iterating through the trie token by token.

```
procedure TRIEBEAMSEARCH(trie, beam_size, LM)
    generations ← [[start token]]
    trie_indices ← [0]
    ongoing_scores ← [0]
    finished_hypothesis ← null
    while |trie_indices| > 0 do
        continuations ← []
        for all hypothesis ← [0, |trie_indices|) do                ▷ Generate candidate continuation for all hypotheses.
            for all new_idx, token ← trie.children(trie_indices[hypothesis]) do    ▷ Continuations coming from trie.
                continuations.append((token, hypothesis, new_idx))
            end for
            if trie.finished_hypothesis_at(trie_indices[hypothesis]) then    ▷ Continuations that finish hypotheses.
                continuations.append((eos token, hypothesis, null))
            end if
        end for
        scores ← LM(generations) + ongoing_scores
        relevant_scores ← []
        for all c ← continuations do
            relevant_scores.append(scores[c[1]][c[0]])
        end for
        score_order ← relevant_scores.argsort()
        reverse(score_order)
        for all idx ← score_order do                    ▷ Update the finished hypothesis.
            if finished_hypothesis ≠ null and finished_hypothesis[1] > score_order[idx] then
                break
            end if
            if continuations[idx][2] = null then                ▷ This must be a finished hypothesis.
                finished_hypothesis ← (generations[continuations[idx][1]] + [continuations[idx][0]], relevant_scores[idx])
            end if
        end for
        new_trie_indices ← []
        new_generations ← []
        new_ongoing_scores ← []
        for all idx ← score_order do
            if |new_generations| = beam_size or finished_hypothesis[1] ≥ score_order[idx] then    ▷ Traditional cut-off
                break
            end if
            if trie_idx ∈ new_trie_indices then    ▷ We have got a cheaper way to tokenize this string.
                continue
            end if
            new_trie_indices.append(continuations[idx][2])
            new_generations.append(generations[continuations[idx][1]] + [continuations[idx][0]])
            new_scores.append(relevant_scores[idx])
        end for
        trie_indices ← new_trie_indices
        generations ← new_generations
        ongoing_scores ← new_ongoing_scores
    end while
    return finished_hypothesis
end procedure
```

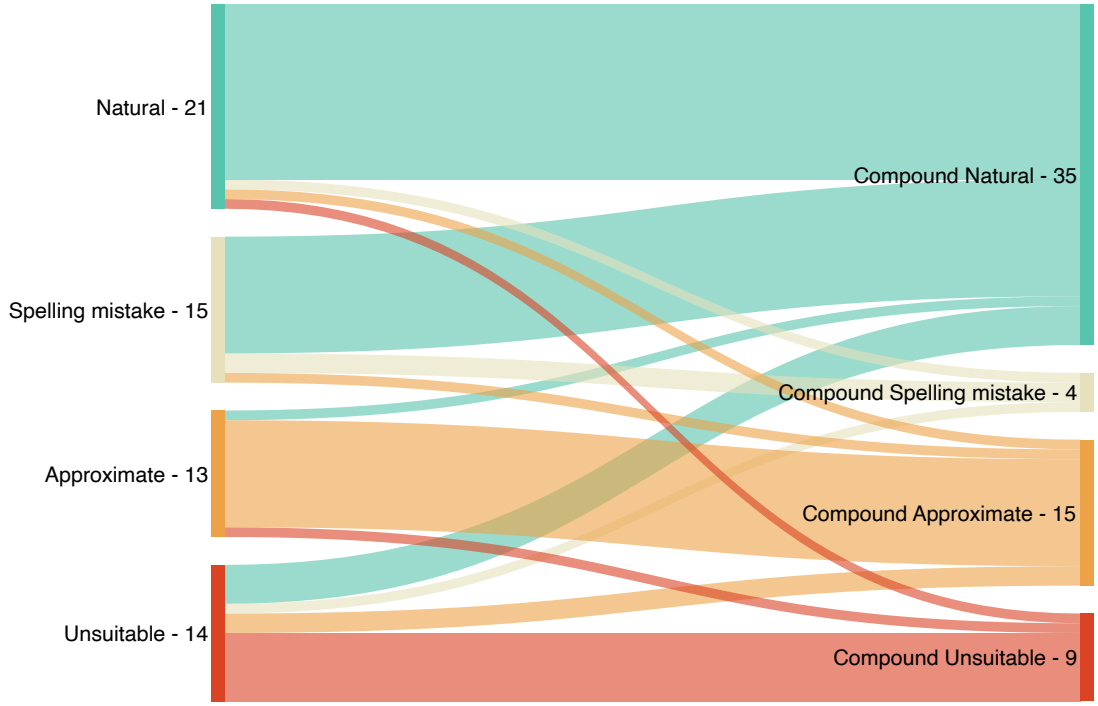

Figure 7: Distribution of the FINELI terms with NMT evaluation on the left and compound translator evaluation on the right.

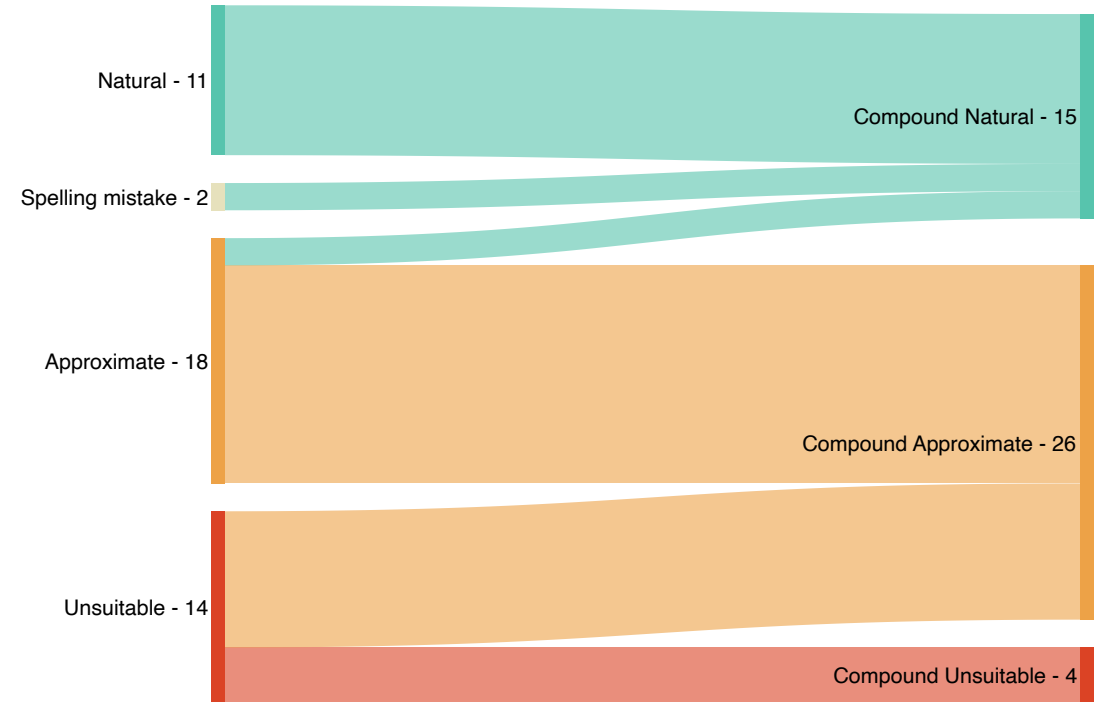

Figure 8: Distribution of the Hydrology terms with NMT evaluation on the left and compound translator evaluation on the right.

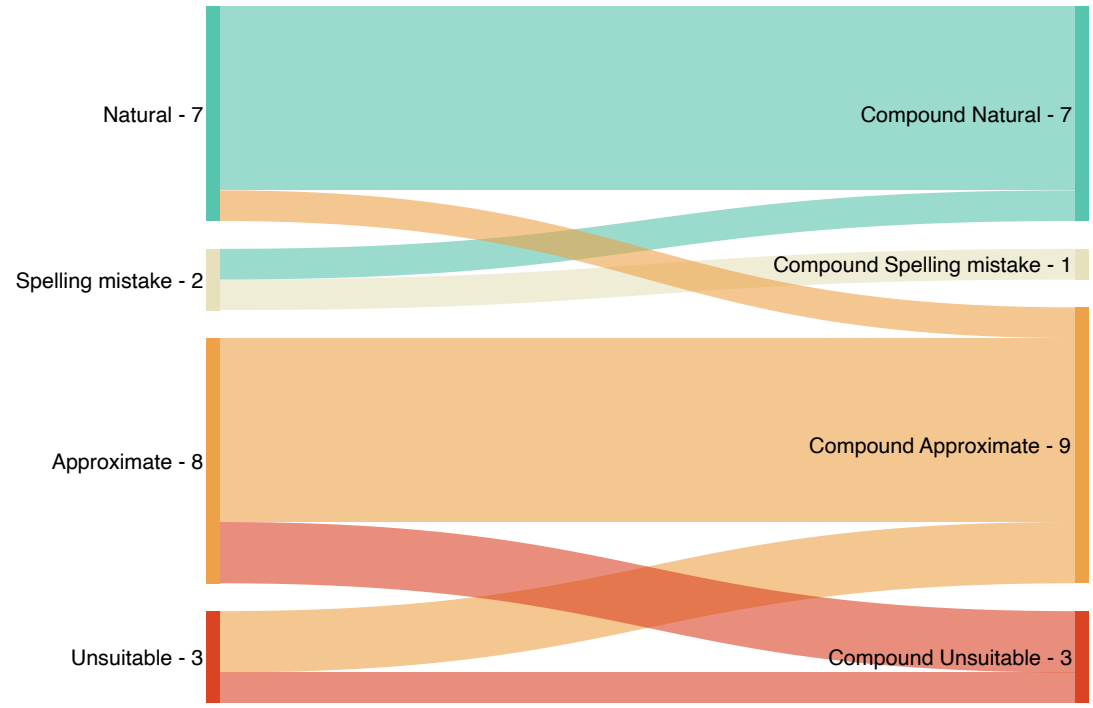

Figure 9: Distribution of the Forest Soil terms with NMT evaluation on the left and compound translator evaluation on the right.

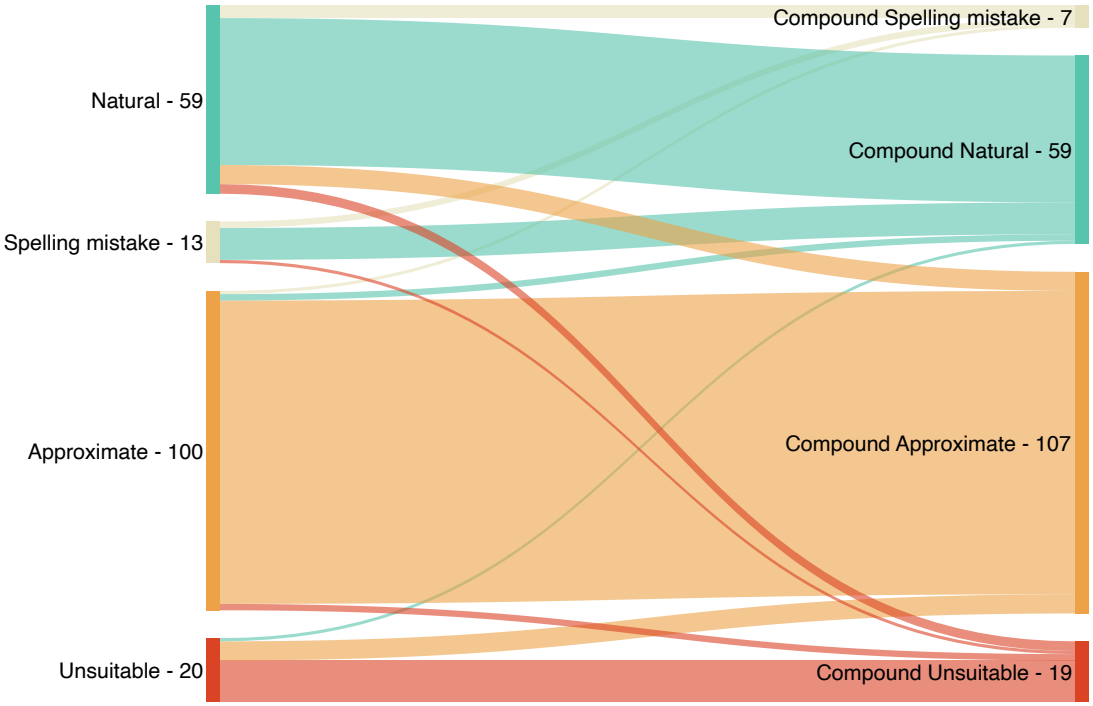

Figure 10: Distribution of the IATE terms with NMT evaluation on the left and compound translator evaluation on the right.