THE UNIVERSITY *of* EDINBURGH

Data Augmentation for Language Generation Inspired by Machine Translation

Pinzhen Chen

Doctor of Philosophy

Institute for Language, Cognition and Computation
School of Informatics
University of Edinburgh

2023

Abstract

The field of natural language processing has witnessed a surge in the adoption of deep learning, which faces notable hurdles when the training data is scarce. This thesis aims to study automatic data augmentation for language generation tasks where acquiring human-annotated data is costly. Drawing from insights in machine translation research, we transfer techniques from this field to a wider array of sequence generation problems. The thesis's initial segment delves into parallel data retrieval for neural machine translation. We devise a method that scores cross-lingual sentences using a translation model itself and approximates pairwise comparisons with trie-constrained decoding. The process does not require document alignment and can identify a reasonable number of parallel sentences. Then, arguing for parallelism between contextualized words and their definitions, we propose to train a unified word-definition modelling system using data augmentation inspired by multilingual translation embeddings. Our system attains superior results for reverse dictionary and definition generation tasks on conventional research datasets and in an international shared task. Finally, we expand generation-based and self-data augmentation to programming language generation tasks including back-translation, monolingual copying, multilingualism, and numeric augmentation. In addition, we attempt to encode numbers as numeric values instead of strings. Significant improvement is observed in code-to-code translation and code-to-text summarization despite starting from powerful language models pre-trained on code and text data.

Lay Summary

Deep learning is a trendy technique that can teach a computer system to understand text and respond. However, the technique needs lots of learning examples, otherwise, it does not lead to satisfactory performance. Employing humans to write such materials is costly, so the thesis explores automatic and inexpensive ways to create learning examples in addition to existing ones. This is named data augmentation. Text translation has received relatively more research, so we hope to adapt it to broader problems revolving around language generation. The thesis starts by looking at systems that can translate, which need to learn from parallel data—sentences in different languages with the same meaning. We use a translation system to translate an input text but force it to consider existing sentences in another language step by step. In this manner, we re-purpose the system to "find" parallel sentences. The second part of the thesis argues that words and their definitions can be viewed as parallel sentences since a word and its definition should have the same meaning. With data augmentation, we build a dual-way dictionary that can be used to find a word given a definition or vice versa. Our work is unique and very competitive compared with other systems. Finally, the thesis attempts to build systems that can deal with programming languages, using data augmentation methods inspired by machine translation research and newly proposed by us. Results show that we can enhance code-to-code translation and code-to-text summarization performance with extra data even when we start from powerful systems that have already ingested many learning examples.

Acknowledgements

I am deeply grateful to those who supported and influenced me throughout my PhD journey. Foremost, I thank my principal supervisor, Kenneth Heafield, for introducing me to the field of machine translation and guiding me in my research endeavours. He has given me much freedom to pursue research directions and attend various events. I also want to extend gratitude to my second supervisor, Barry Haddow, for his support and guidance, particularly during my final year. I appreciate his thorough, reviewer-like feedback on paper drafts.

The internal examiner of this thesis, Alexandra Birch, was extremely encouraging and kind. My external examiner Ondřej Bojar offered insightful feedback and questions. Walid Magdy attended my first-year review and encouraged the use of pre-trained models. I thank Nikolay Bogoychev for his help with tries and Marian-NMT, for inviting me to an Easter egg fight, and for sharing the highs and lows of side projects. Zheng Zhao assisted with and proofread the work on word-definition modelling. Gerasimos Lampouras and Ignacio Iacobacci hosted an internship on code generation, which contributed to part of this thesis.

I enjoyed academic discussions and collaborations with many people, including Laurie Burchell, Zhicheng Guo, Hanxu Hu, Vivek Iyyer, Shaoxiong Ji, Faheem Kirefu, Tsz Kin Lam, Yuanchao Li, Jinhong Lu, Nikita Moghe, Proyag Pal, Ashok Urlana, Shuzhuang Xu, Yijun Yang, Simon Yu, Biao Zhang, Zeyu Zhao, Dawei Zhu, and Wenhao Zhu. I also thank colleagues in the StatMT group for inspiring conversations and engaging reading sessions. Additionally, I appreciate the professional services staff in the School of Informatics for creating a wonderful working environment.

On a personal note, I am deeply indebted to my parents for their unwavering support. I also thank all my friends, both near and far, for the time spent together, the meals, chats, and trips that provided much-needed breaks and laughter. Special thanks to Jingyi for her care and encouragement.

Finally, I wish to acknowledge the following projects for funding my research: ParaCrawl, MATERIAL, Bergamot, and HPLT.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Pinzhen Chen*)

Jingyi, will you marry me?

Table of Contents

Chapter 1

Introduction

1.1 Motivation

In recent years, there has been a rapid increase in the use of deep learning techniques for natural language processing (NLP), leading to significant advances in a wide range of tasks. A notable distinction between deep learning-based systems and their predecessors is the heavy reliance on a larger training dataset (LeCun et al., 2015). Data scarcity poses a significant challenge for neural methods (Hedderich et al., 2021; Li et al., 2022) and has been referred to as one of the "4 biggest open problems in NLP"[1] and one of the "six challenges for neural machine translation" (Koehn and Knowles, 2017). Although the recently emerged pre-trained language models (PLMs) have ingested tonnes of text and exhibited potential for cross-lingual transfer (Devlin et al., 2019; Conneau et al., 2020), it has been found that data augmentation is still helpful in several NLP tasks on top of PLMs by Şahin (2022) and later by us (Chen and Lampouras, 2023) in this thesis. Furthermore, low-resource conditions are not merely a result of the language but could also be due to the nature of a task or domain specificity, which PLMs might not easily solve. Thus, data deficiency remains a persistent barrier, which encourages research on data acquisition and augmentation in low-resource situations (Feng et al., 2021).

Language generation systems, which produce a text sequence in response to input, such as machine translation (Sutskever et al., 2014), text summarization (Nallapati et al., 2016), and definition modelling (Noraset et al., 2017), can hardly avoid data scarcity in many cases (Howcroft and Gkatzia, 2022). Employing human annotators to create such data is prohibitively expensive, especially when long input-output

[1] https://www.ruder.io/4-biggest-open-problems-in-nlp/

sentences need to be created as opposed to single-word labels in text classification. Therefore, this thesis focuses on developing automatic methods to increase the data availability for sequence-to-sequence generation. Although a few surveys state that data augmentation is still "relatively under-explored" (Feng et al., 2021; Howcroft and Gkatzia, 2022), we observe effective and readily transferable data practices in the field of neural machine translation—a typical and widely investigated generation problem (Sennrich et al., 2016a; Fadaee et al., 2017; Currey et al., 2017; Johnson et al., 2017). Hence, it is unsurprising to witness works derived from machine translation research being applied to various other NLP problems (Junczys-Dowmunt et al., 2018b; Zhang and Duh, 2021; Chen and Heafield, 2022). Likewise, our thesis seeks to apply machine translation-inspired techniques to broader sequence generation tasks.

1.2 Research Scope

The first part of the thesis is grounded on the data extraction efforts in the ParaCrawl project (Bañón et al., 2020), which mines parallel sentences from web crawls to acquire data for machine translation. It first aligns web pages (also denoted as documents) in different languages using their domain information as a heuristic and then looks for sentence alignments within document pairs (Uszkoreit et al., 2010). However, due to distinct crawling behaviours, raw data might not necessarily be grouped and indexed by web domains, which hinders document alignment. Hence, we are interested in **finding a way to reduce the dependence on document alignment for parallel data retrieval**. Building on parallel sentence filtering (Junczys-Dowmunt, 2018) and constrained decoding (Hokamp and Liu, 2017; Post and Vilar, 2018), we create a standalone sentence aligner modified from a standard translation system for automatic sentence pair scoring and retrieval.

Further, inspired by the ideas of obtaining multilingual sentence embeddings from hidden states in neural translation models (Artetxe and Schwenk, 2019a,b) we learned in the tasks of parallel sentence mining, we study **whether similar representation learning with augmented data can help word-definition modelling tasks**: reverse dictionary and definition generation (Hill et al., 2016a; Noraset et al., 2017; Bosc and Vincent, 2018). We argue that a contextualized word and its definition can be philosophically viewed as "parallel"—they convey the same meaning but have different surface forms. Along with a multi-way self-data augmentation approach, we design a unified model that utilizes the enriched data to learn both tasks simultaneously.

Finally, while data augmentation for machine translation has been extensively studied, observe an absence of similar undertakings for the emerging generation tasks involving programming languages. Code generation problems can be comparable to text translation in terms of their sequence-to-sequence nature, for instance, code-to-code translation and code-to-text summarization (Lu et al., 2021). Therefore, we examine whether we can **adapt data augmentation approaches from text translation to the code domain**: back-translation, copying, multilingualism, and word replacement. Moreover, given the importance of numeric correctness in code, we also suggest a novel way to encode numbers directly as numeric values instead of string tokens. Given the swift progress in PLMs for programming languages, another intriguing question is **whether code data augmentation techniques are beneficial for PLM fine-tuning**, which receives differing observations in Feng et al. (2021) and Şahin (2022)'s surveys.

Since there is no consensus on taxonomy (Feng et al., 2021; Hedderich et al., 2021; Şahin, 2022), we classify the automatic data augmentation methods this thesis examines into three categories. The first is **retrieval-based augmentation**. It automatically extracts supervised data from a pool of unlabelled data originally unusable for training. This is applicable when a considerable amount of resources need to be processed into task-specific data. With a complex pipeline, the resulting data is usually of satisfactory quality and size. We also study **self-data augmentation** which is a data modelling enhancement method without reliance on any external resources. The essence is to leverage existing training data to create larger resources that are better suited for purposes. Typical approaches include noising, autoencoding and multilingualism. Such low-cost augmented training yields effective and robust systems (Wei and Zou, 2019) and enables multi-tasking (Sánchez-Cartagena et al., 2021). The final part of the thesis employs several **generation-based augmentation** techniques. With seed resources like models or handcrafted rules, additional data can be synthesized from either labelled or unlabelled data. Despite often being lower in quality, the synthesized data can cover more input variations and incorporate external knowledge.

1.3 Thesis Structure and Contribution

This thesis uses the pronoun "we" as a customary practice in the field. It is essential to clarify that the scientific work has been conducted by the author of this thesis. Individual contributions are stated in Section 1.4.

Chapter 2 As a background chapter, it explains sequence-to-sequence generation, data processing, and system evaluation. It then outlines the Transformer architecture which is used throughout the thesis. This is followed by an introduction of data augmentation from three perspectives depending on the reliance on external resources: retrieval-based augmentation, self-data augmentation, and generation-based augmentation.

Chapter 3 This chapter presents our research on a sentence alignment method that does not require a document aligner in the first place. We utilize a translation model to score the parallelism between cross-lingual sentences and approximate source-target pairwise comparisons with trie-constrained decoding. It presents a novel idea of repurposing translation decoding for data retrieval which conducts similarity scoring and efficient comparison simultaneously. The method is benchmarked on parallel sentence extraction from comparable corpora and compared with recent works. The author's contribution is as follows:

- We proposed an automatic data retrieval method for machine translation and carried out experiments on a popular benchmark.
- We discovered defects in a popular dataset used by many parallel sentence mining research papers and reported this to the task organizer and data creator.
- We analyzed the limitations of our method in terms of scalability and directionality with reference to other parallel sentence mining strategies.

Chapter 4 We put forward a unified word-definition modelling approach trained on multi-way data enhancement following the concept of multilingual sentence embeddings. The trained model can be viewed as a dual-way neural dictionary that supports word retrieval given a definition and vice versa. We test the model in various scenarios: languages, embeddings, and data availability. The contribution includes:

- We adapted a self-contained data augmentation approach to reverse dictionary and definition generation.
- We devised a unified model to make use of the data to learn word retrieval and definition generation simultaneously.
- We attained superior results for both tasks through automatic metrics and human evaluation. We also signalled the difficulty in evaluating definition generation.

- We performed analysis across different embedding architectures, languages, and linguistic features.

In addition, our participation in SemEval 2022 Task 1 won multiple tracks. Our description paper was awarded a best paper honourable mention out of 213 papers for "advancing understanding of a problem and available solutions, and having a strong analysis component in the evaluation, as well as a clear and reproducible description of the problem, algorithms, and methodology".[2]

Chapter 5 It describes our efforts in exploring data augmentation approaches for newly emerged research tasks at the intersection of programming and natural language generation: code-to-code translation, code-to-text summarization, and text-to-code synthesis. Our methodology includes generation-based augmentation and self-augmentation inspired by machine translation literature, as well as a novel numeric-aware scheme. We examine our strategies in the fine-tuning stage of code PLMs. We summarize our contribution as:

- We made one of the earliest attempts in data augmentation for three code generation tasks.
- We achieved favourable performance in code translation and summarization and provided insights on room for improvement in code synthesis.
- We conducted manual inspections to reveal test set imperfections and the benefits from additional data: improvement in the code style and numeric consistency.
- We demonstrated that data augmentation can work on top of PLMs for generation problems concerning programming languages.

Chapter 6 We summarize our research findings and suggest avenues for future work.

1.4 Publications

Content chapters in the thesis have been developed from the following publications, where we also list the contribution of each author:

[2] https://semeval.github.io/SemEval2022/awards

- Chapter 3 is based on *Parallel Sentence Mining by Constrained Decoding*, by Pinzhen Chen, Nikolay Bogoychev, Kenneth Heafield, and Faheem Kirefu, in Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (Chen et al., 2020b). Chen conceptualized the idea, ran experiments and analysis, and wrote the paper. Bogoychev implemented the decoding algorithm and helped with paper writing. Heafield supervised the work and edited the paper. Kirefu provided Russian and French translation models as well as a setup for a pilot experiment not included in the paper.

- First half of Chapter 4 is based on *A Unified Model for Reverse Dictionary and Definition Modelling*, by Pinzhen Chen and Zheng Zhao, in Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Chen and Zhao, 2022b). Chen conceptualized the idea, implemented the model, conducted experiments and analysis, and wrote the paper. Zhao processed data and helped with model training and human evaluation.

- Second half of Chapter 4 is based on *Edinburgh at SemEval-2022 Task 1: Jointly Fishing for Word Embeddings and Definitions*, by Pinzhen Chen and Zheng Zhao, in Proceedings of the 16th International Workshop on Semantic Evaluation (Chen and Zhao, 2022a). Chen ran experiments, conducted analysis, and wrote the paper. Zheng helped with paper writing.

- Chapter 5 is based on *Exploring Data Augmentation for Code Generation Tasks* by Pinzhen Chen and Gerasimos Lampouras, in Findings of the Association for Computational Linguistics: EACL 2023 (Chen and Lampouras, 2023). Chen conceptualized the idea, conducted experiments and analysis, and wrote the paper. Lampouras supervised the work and edited the paper.

We also acted as the first or co-first author on some papers related to the thesis:

- *Approaching Neural Chinese Word Segmentation as a Low-Resource Machine Translation Task*, by Pinzhen Chen and Kenneth Heafield, in Proceedings of the 36th Pacific Asia Conference on Language, Information and Computation (Chen and Heafield, 2022). We formalized Chinese word segmentation, conventionally a sequence tagging problem, as a generation task and took inspiration from low-resource practices in neural machine translation.

- *PMIndiaSum: Multilingual and Cross-lingual Headline Summarization for Languages in India*, by Ashok Urlana, Pinzhen Chen, Zheng Zhao, Shay Cohen, Manish Shrivastava, and Barry Haddow, in Findings of the Association for Computational Linguistics: EMNLP 2023 (Urlana et al., 2023). We used a parallel sentence extraction pipeline to deliver an end-to-end multilingual and cross-lingual summarization dataset to substitute translate-and-summarize approaches.

- *Monolingual or Multilingual Instruction Tuning: Which Makes a Better Alpaca*, by Pinzhen Chen, Shaoxiong Ji, Nikolay Bogoychev, Andrey Kutuzov, Barry Haddow, and Kenneth Heafield, in Findings of the Association for Computational Linguistics: EACL 2024 (Chen et al., 2024). We compared chat-like instruction tuning on monolingual and multilingual data augmented by machine translation.

Chapter 2

Background Literature

2.1 Sequence-to-Sequence Generation

This thesis deals with language generation tasks that can be formalized as sequence-to-sequence modelling. By defining an input sequence $X = \left[x_1, x_2, \ldots, x_{|X|}\right]$, $x_i \in V_X$ which is a vocabulary set for X, and an output sequence $Y = \left[y_1, y_2, \ldots, y_{|Y|}\right]$, $y_i \in V_Y$ which is a vocabulary set for Y, we can formulate the generation of Y from a model θ as modelling the conditional probability of Y given X at both training and inference time:

$$P(Y|X;\theta)$$

In deep learning, an artificial neural network is used to take an input and approximate the above probability of an output. It is composed of often multiple layers of neurons, where each neuron takes a vector input vector and produces a scalar output. A neuron computes a weighted sum of the input vector, optionally adds a bias term, and then usually applies a non-linear activation function. All neuron weights and biases are referred to as model parameters.

2.1.1 Training

Training, also called "learning", is the process of updating a model's parameters using some kind of stimulus—in our thesis, training data. Supervised training refers to when the model is provided with both the input and ground truth of that input. A *ground truth* is regarded as the correct answer to the input for a particular purpose, and it can also be called a *reference* or a *gold* output. A neural network model learns to produce outputs that are as close as possible to the ground truth given an input. One

popular way to achieve this is to maximize the likelihood of the ground truth output $Y^* = \left[y_1^*, y_2^*, \ldots, y_{|Y^*|}^*\right]$ given X by adjusting the model parameters θ:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} P(Y^*|X;\theta)$$

Maximizing the likelihood of the ground truth is equivalent to minimizing its negative log-likelihood. The latter is conventionally used as a loss $\mathcal{L}$ to be minimized during neural network training. It is a function of the ground truth as well as the model prediction given the input and model parameters. One way to compute this is to add the negative log-likelihood of each token in the ground truth sequentially—assuming autoregressive generation, whereby tokens are generated one by one. The addition of step-wise log probabilities brings in two advantages: it prevents underflow when probabilities are multiplied and addition is often faster to perform than multiplication. This formulation can be further interpreted as a multi-label cross-entropy loss with only one label being the gold. Each label corresponds to an item v in vocabulary V_Y which the model can choose from at each step. We use $I()$ to denote a Boolean function that maps to $\mathbb{Z}_2 = \{0,1\}$. The above explanation corresponds to the expansion of loss function $\mathcal{L}$ below:

$$\begin{aligned}
\mathcal{L}(X,Y^*;\theta) &= -\log P(Y^*|X;\theta) \\
&= -\log \prod_{t=1}^{|Y^*|} P(y_t^*|y_{<t}^*,X;\theta) \\
&= -\sum_{t=1}^{|Y^*|} \log P(y_t^*|y_{<t}^*,X;\theta) \\
&= -\sum_{t=1}^{|Y^*|} \sum_{v \in V_Y} I(v = y_t^*) \log P(v|y_{<t}^*,X;\theta)
\end{aligned}$$

$I(v = y_t^*)$ only evaluates to 1 when the generated token is the ground truth token at time step t and to 0 otherwise. Since the training data might contain noise and mistakes, and human languages' variability allows for different generations to be valid, neural network training usually adopts a regularization technique named *label smoothing* (Szegedy et al., 2016). It removes a small probability mass $\delta \in [0,1)$ from the ground truth token and assigns $\frac{\delta}{|V|-1}$ to each other token in the vocabulary. This can be achieved by replacing $I(y_t^* = v)$ with $I(y_t^* = v)(1-\delta) + I(y_t^* \neq v)\frac{\delta}{|V|-1}$ in the loss function. Another common regularization technique used in our model training is *dropout* (Srivastava et al., 2014). During training, without a small probability p, it randomly

ignores the activation outputs from some neurons in a neural network and scales the outputs from other neurons to $\frac{1}{1-p}$, which prevents the co-adaptation of several weights to reduce model overfitting. Our model training uses the two techniques.

To achieve generalization to unseen data, the training process adjusts the model parameters to reduce the loss on a sizable collection of input-output pairs referred to as training data $D_{train} = \left\{(X_1, Y_1^*), (X_2, Y_2^*), \ldots, (X_{|D|}, Y_{|D|}^*)\right\}$. Optimizers manage the adjustment of parameters using backpropagation, for example, by controlling the learning rate and gradients. Ideally, they aim to find the best set of parameters that minimize the overall loss across all training data Among many sophisticated ones, the Adam optimizer (Kingma and Ba, 2015) is a prevalent choice. Since optimizers are not the subject of our research, we adopt this throughout.

2.1.2 Inference

The procedure of generating an output in the field of natural language processing is referred to as *inference* or *decoding*. At this stage, the parameters will not change, and the model produces an output after reading an input. In our thesis, we only use *autoregressive decoding*, which means that a model produces the output words one by one, usually in the same order as how the language is written or spoken. It is the default in sequence generation tasks. Although there exist non-autoregressive decoding options where multiple tokens can be created at the same time for speed consideration, the quality is usually not as good as autoregressive decoding.

The goal of inference is to find the best possible output as opposed to an arbitrary one. The search process can thus be expressed as looking for the $\hat{Y}$, among all possible output Y's, that is associated with the highest likelihood given the input X and the model θ. This can further be expanded as a chained probability given autoregressive generation, with $t \geq 0$ denoting the time step of generation:

$$\begin{aligned}\hat{Y} &= \operatorname*{argmax}_{Y} P(Y|X;\theta) \\ &= \operatorname*{argmax}_{Y \in \mathbb{Y}} \prod_{t=1}^{|Y|} P(y_t|y_{<t}, X;\theta)\end{aligned}$$

Here we use $\mathbb{Y}$ to represent the search space of all possible output Y's. However, this space grows exponentially with regard to the allowed length of Y's—it has a size of $|V|^{|Y|}$. This makes it intractable to enumerate all possible outputs to find the optimal output according to the model probability. A simplistic solution, named *greedy search*,

is to select the token having the highest conditional probability at each step t. It approximates the global optimal solution by selecting the step-wise optimal:

$$\begin{aligned}\hat{Y} &= \underset{Y \in \mathbb{Y}}{\operatorname{argmax}} \prod_{t=1}^{|Y|} P(y_t | y_{<t}, X; \theta) \\ &\approx \prod_{t=1}^{|Y|} \underset{y_t}{\operatorname{argmax}} P(y_t | y_{<t}, X; \theta)\end{aligned}$$

Greedy search is easy to implement and fast to compute, but it is prone to errors because it picks the local best without considering a longer context into the future steps. Hence, a more popular option in the field is *beam search*, which keeps a few candidates over a wider context. Beam search with a beam size B can keep the top B hypotheses at each time step t instead of greedily choosing the best. The beam size B is an adjustable hyperparameter that controls the trade-off between efficiency and search accuracy. Beam search is equivalent to greedy search when the beam size is set to 1. Assuming a maximum allowed generation length L and a vocabulary V to pick tokens from, the algorithm can be described using pseudo-code in Algorithm 1.

Algorithm 1 Beam search: maximum output length L, beam size B, vocabulary V.

```
beam_0 ← {<s>}
completed ← {}
for time step t in 1 to L do
    beam_t ← {}
    for hypothesis h in beam_{t-1} do
        beam_t ← beam_t ∪ Continue(h, V)
    beam_t ← NBest(beam_t, B − |completed|)
    Move hypotheses ending with </s> from beam_t to completed.
    if beam_t is empty then
        return Best(completed)
return Best(completed)
```

We detail the beam search algorithm by referring to Algorithm 1. First, the beam is initialized with a list containing a single hypothesis that is a begin-of-sentence token (<s>), as well as an empty "completed" list that will hold completed hypotheses. Next, at each time step, the beam is expanded by appending all possible tokens in the vocabulary V to each hypothesis in the beam. The expanded beam is then pruned

to only keep the top $B - |\text{completed}|$ hypotheses by probabilities determined by the model. Once a hypothesis in the beam ends with an end-of-sentence token (</s>), it is moved to the completed list. The algorithm terminates if the maximum length is reached or the completed list grows to beam size B. The final search outcome is the hypothesis with the best probability in the completed list. Based on this algorithm, Chapter 3 develops constrained beam search to facilitate the generative retrieval of translation data.

2.1.3 Text processing

Neurons in a neural network take a vector as input, so text data cannot be directly fed into a neural network without being converted into numerical vectors. This section introduces the two processing steps that are common in the field: tokenization which splits a text string into individual units called tokens as well as word embedding which then converts each token into a vector.

Tokenization An input string needs to be converted to a set of vectors to form an input to a neural network. First, a tokenizer converts an input sequence X into discrete tokens $X = \left[t_1, t_2, \ldots, t_{|L|}\right]$ with length L using a vocabulary set V, where $t_i \in V, \forall i = 1, 2, \ldots, |L|$. Modern tokenizers usually run sub-word strategies to determine a vocabulary of tokens, to simultaneously control the granularity and vocabulary size as well as reduce unknown word appearances (Sennrich et al., 2016b; Kudo, 2018). In practice, each item in the vocabulary set V is assigned a unique number, namely vocabulary index, between 1 and $|V|$ inclusive. Accordingly, the tokenized input string can be converted to a sequence of indices $X = \left[i_1, i_2, \ldots, i_{|L|}\right]$ by substituting tokens with their indices.

Word embeddings Collectively, word embeddings form entries in a learnable matrix $E_{word} \in \mathbb{R}^{|V| \times d}$ with hidden dimension d to convert vocabulary items into vectors of size d that are readable by a neural network. The word embedding matrix usually serves as a lookup table, where the i-th vector entry corresponds to the word embedding for the vocabulary item associated with index i, $\forall i = 1, 2, \ldots, |V|$. With the embedding matrix, a sequence of input token indices can be further transformed into a sequence of word embeddings $X = \left[e_1, e_2, \ldots, e_{|L|}\right]$.

2.1.4 Evaluation and metrics

A trained model can be evaluated by measuring its performance on a test set that has not been exposed to the model during training. A test set D_{test} normally comprises sufficient samples of inputs X_{test} and ground truths Y^*_{test} that are high-quality and representative of a specific interest. After we perform inference on each test input to obtain respective model output $\hat{Y}_{test}$, we can compute a metric function of the outputs and at least one of the inputs or ground truths to evaluate the model's performance. There are two popular paradigms in model evaluation. First, evaluation using automatic metrics, which often compares the model generations with the ground truths to derive a score that is designed to reflect the correctness of the output. An alternative is to employ human evaluators to judge the quality of the model outputs.

String-based metrics String-based metrics refer to those that measure the similarity between a generated output and the ground truth by examining their surface forms, e.g. string overlap and edit distance. These metrics offer the advantage of being inexpensive and quick to compute. Common metrics for text generation include BLEU (Papineni et al., 2002) and ROUGE (Lin, 2004); these two with their variants are extensively used in this thesis. We also introduce and report scores from other specialized metrics in the respective chapters later.

BLEU is a precision-based metric that measures the percentage of n-grams in the model prediction that also appear in the ground truth. The score is first computed as the geometric mean of n-gram precisions from 1 to n. If the generation length is shorter than that of gold, the score is multiplied by the generation length divided by the ground truth length. The multiplier acts as a length penalty if the generation is too short and thus receives an undesirably high precision score. Conventionally, n is set to 4 to consider fluency in addition to adequacy, and this particular version is sometimes referred to as BLEU-4.

ROUGE has several variants that compute precision, recall, and F1 in different fashions. Our work adopts ROUGE-n and ROUGE-L. The former derives an F1 score from the n-gram precision and recall between the model output and the ground truth. The latter, ROUGE-L, first identifies the longest common subsequence—words that are shared, ordered, but not necessarily consecutive—between an output and the reference. Then, precision or recall is calculated by dividing that particular subsequence's length by the output's length or the reference's length respectively.

Neural network-based metrics Recent research in automatic metrics for language generation started to embrace neural representations. Metrics that operate on neural string representations enable better semantic comparison rather than focusing on surface forms. Many metrics in this category calculate distance scores between aligned contextualized word representations from pre-trained language models (Zhao et al., 2019; Zhang et al., 2020). Specifically, we adopt MoverScore which uses Word Mover's Distance as a distance function. Some learned neural metrics take a step further by learning a scoring model that directly takes sentence-level representations obtained from pooling word embeddings as inputs (Shimanaka et al., 2018; Rei et al., 2020). Neural metrics can correlate with human judgements better, but they are more computationally expensive and less interpretable. Also, these metrics may be susceptible to domain shifts when the data being evaluated is different from the backbone model's training data.

Human evaluation Human evaluation is expensive and time-consuming but deemed to be reliable, as it with no doubt reflects the human perspective on the generation quality. It is usually carried out by presenting human participants with the model output and the input (source-based), or alternatively the ground truth instead of the input (reference-based), and asking them to judge the quality of the model output. The judgement process can be done in various ways, e.g. rating outputs on a fixed scale or choosing the best output among a few candidates from different systems. The model-level performance can be measured by aggregating the scores of all outputs in a test set, or by counting the number of times a certain system is preferred. We use human evaluation in situations where automatic metrics are not representative, for example, when a problem is too difficult or when they cannot differentiate several systems.

2.2 Transformer

All our experiments use the Transformer model (Vaswani et al., 2017), which currently is a popular and effective neural network architecture to parameterize $P(Y|X;\theta)$. We include an illustration of the model in Figure 2.1. In this architecture, the relationship between words, or more precisely, between word representations, is modelled solely by an *attention* mechanism. Given its outstanding performance, the model has become the default in many natural language processing tasks. We start with a close look at the attention mechanism and then describe the Transformer model structure.

2.2.1 Attention

Key, query, and value Diving deeper into the Transformer model, the most iconic module is its attention layers. In each attention layer, for a representation vector $x \in \mathbb{R}^d$ with hidden dimension d, three trainable weight matrices $W_q \in \mathbb{R}^{d \times d_q}$, $W_k \in \mathbb{R}^{d \times d_k}$, $W_v \in \mathbb{R}^{d \times d_v}$ convert it into query Q_x, key K_x, and value V_x, with d_q, d_k, and d_v denoting the dimensions of the resulting key, query, and value respectively:

$$Q_x = xW_q, \quad K_x = xW_k, \quad V_x = xW_v$$

In practical implementations, a sequence of such vectors $x_1, x_2, \ldots, x_L$, with L denoting the sequence length, are usually packed as a matrix $X \in \mathbb{R}^{L \times d}$. Consequently, the resulting queries, keys, and values can be represented as matrices $Q_X \in \mathbb{R}^{L \times d_q}$, $K_X \in \mathbb{R}^{L \times d_k}$, and $V_X \in \mathbb{R}^{L \times d_v}$ respectively.

Scaled dot-product attention In natural language processing, the vectors mentioned above correspond to the word (token) embeddings of tokens and the matrix corresponds to the entire sequence. For each token, the attention mechanism computes similarity scores between this token and all tokens in the sequence including itself. The score is determined by comparing the query of the former and the key of the latter. Such scores are also referred to as attention, and a model uses these scores to refine the representation for each token, by aggregating all tokens' representations in the same sequence. The intuition behind this can be traced back to John Rupert Firth's distributional semantics hypothesis—"a word is characterized by the company it keeps".

To illustrate how attention is computed between two sequences or on a sequence itself, without loss of generality, we define another sequence $Y = [y_1, y_2, \ldots, y_M]$ with length M. The attention from each token x_i to each token y_j, where $i = 0, 1, \ldots, L$ and $j = 0, 1, \ldots, M$, can be written as $\alpha_{x_i, y_j} = Att(Q_{x_i}, K_{y_j}) \in \mathbb{R}$. At the sequence level, the attention from X to Y can be written as $\alpha_{X,Y} = Att(Q_X, K_Y) \in \mathbb{R}^{L \times M}$. While $Att()$ can be implemented in various ways, for instance, as cosine similarity or as a trainable neural network (Graves et al., 2014; Bahdanau et al., 2015), the Transformer work uses a dot product scaled by the square root of the size of the key vector being attended to:

$$\alpha_{X,Y} = \frac{Q_X K_Y^T}{\sqrt{d_{k_y}}}$$

Note that this requires the query of the attending token vector and the key of the attended token vector to have the same size: $d_{q_x} = d_{k_y}$.

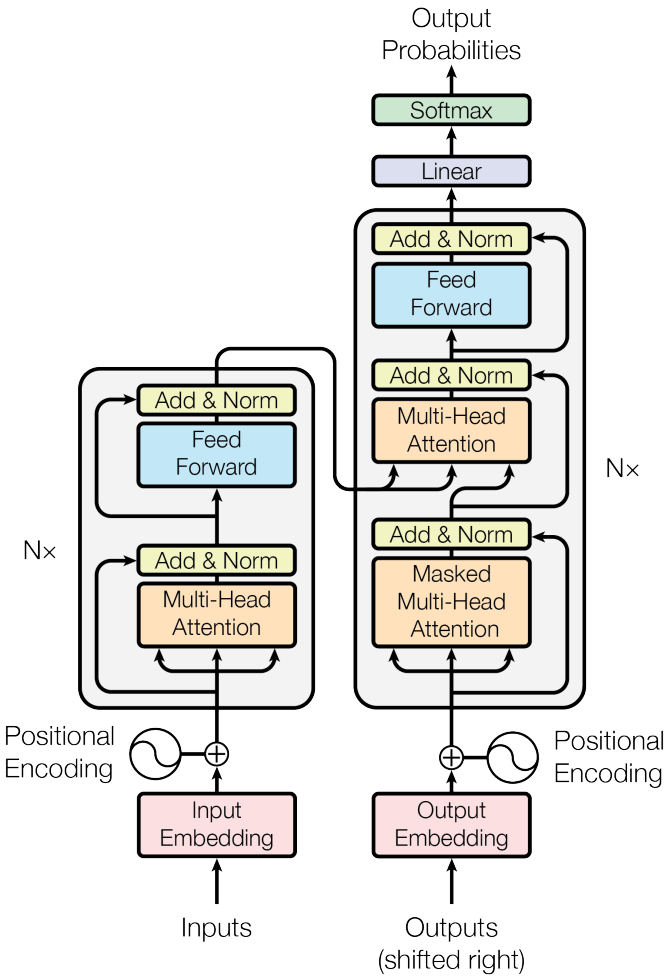


Figure 2.1: The Transformer model illustrated by Vaswani et al. (2017).

In some cases, attention from elements in X to certain tokens in Y is not wanted, for example, when at inference time the tokens in the future steps will have not been generated. We use a mask matrix $M \in \{0, -\infty\}^{L \times M}$ to retain desired attention scores but bring those unwanted to negative infinity. Then the attention scores go through a softmax function along Y, to give an attention matrix:

$$A = \text{softmax}(\alpha_{X,Y} + M)$$

The softmax computation converts each X token's attention scores over all tokens in Y into a distribution that sums to 1. It is a form of normalization that suppresses too large or tiny values to between 0 and 1; it also provides some interpretability because it can be seen as a probability distribution of each X token's attention over all tokens in Y. Effectively, the negative infinity scores forced by the mask matrix now will have

a normalized score of 0.

Finally, the attention matrix is multiplied by the value matrix of Y: $AV_Y \in \mathbb{R}^{L \times d_{v_y}}$. This step can be interpreted as aggregating information from all tokens in Y as a context, based on the similarity between a query of a single token in X and the keys of all Y tokens. With trainable key, query, and value weight matrices, the scaled dot product attention can be expressed as:

$$\begin{aligned}\text{ScaledDotProductAtt}(X,Y) &= \text{softmax}\left(\frac{Q_X K_Y^T}{\sqrt{d_{k_y}}} + M\right) V_Y \\ &= \text{softmax}\left(\frac{XW_{q_x}(YW_{k_y})^T}{\sqrt{d_{k_y}}} + M\right) YW_{v_y} \in \mathbb{R}^{L \times d_{v_y}}\end{aligned}$$

Multi-head attention Vaswani et al. (2017) gave the scaled dot-product attention operation the name *head*. Instead of performing attention once, they developed *multi-head attention* as shown in Figure 2.1, where sequences X and Y are converted into different queries, keys, and values using different weight matrices, to perform the scaled dot-product attention as many as H times. The results from H heads, with dimensions $d_{head_1}, d_{head_2}, \ldots, d_{head_H}$, are concatenated then projected to the hidden dimension d using a weight matrix $W_o \in \mathbb{R}^{(d_{head_1}+d_{head_2}+\ldots+d_{head_H}) \times d}$:

$$\begin{aligned}\text{MultiHeadAtt}(X,Y) = \text{concat}(&\text{ScaledDotProductAtt}_1(X,Y), \\ &\text{ScaledDotProductAtt}_2(X,Y), \\ &\ldots, \\ &\text{ScaledDotProductAtt}_H(X,Y))W_o\end{aligned}$$

where each $\text{ScaledDotProductAtt}_h(X,Y)$ with $h = 1,2,\ldots,H$, uses a different set of query, key, and value weight matrices $W_{q_x,h}$, $W_{k_y,h}$, and $W_{v_y,h}$ to transform X and Y. Also, the Transformer paper makes the queries, keys, and values equal in size for all H heads:

$$d_q = d_k = d_v = d_{head_1} = d_{head_2} = \ldots = d_{head_H} = \frac{d}{H}$$

Such multi-head attention gives a model multiple chances to transform the input vectors, and different heads potentially learn different information with varying importance (Raganato and Tiedemann, 2018; Voita et al., 2019). This is applied to sequences in three places: encoder-to-encoder (between inputs and inputs), decoder-to-decoder (between outputs and outputs), as well as encoder-to-decoder (between inputs and outputs). It is therefore permissible and necessary to have $X = Y$ in the first two cases; such attention is also called self-attention.

2.2.2 Positional embeddings

The Transformer architecture encodes all sequence tokens in a parallel manner and the attention outcomes are solely based on the input embeddings (vectors). Thus the model is not supplied with explicit information about word ordering. This leads to the problem that changing the input token positions will result in the same output representations, which is undesirable because natural language texts are ordered. As such, the Transformer model is provided with explicit word position information via the extra *positional embeddings*. The particular scheme uses alternating sine and cosine functions to compute the numerical values to form an embedding for a position.

Formally, for each position i in a sequence of word embeddings $X = \left[e_1, e_2, \ldots, e_{|L|}\right]$, a positional embedding $pe_i \in \mathbb{R}^d$ is constructed. The value at each position j of the positional embedding, where $j = 1, 2, \ldots, d$, is computed as:

$$pe_i[j] = \begin{cases} \sin\left(\frac{i}{10000^{\frac{j-1}{d}}}\right), & \text{if } j(\bmod 2) = 1 \\ \cos\left(\frac{i}{10000^{\frac{j-2}{d}}}\right), & \text{otherwise} \end{cases}$$

The use of sinusoidal functions with different wavelengths could help the model to recognize the relative distance between two absolute positions in addition. Another benefit is that it allows easy extrapolation to sequences longer than the maximum length in training. The final input embeddings to the Transformer model are the position-wise additions of word embeddings and positional embeddings:

$$X = \left[e_1 + pe_1, e_2 + pe_2, \ldots, e_{|L|} + pe_{|L|}\right]$$

Nonetheless, some later Transformer-based models opted for learned positional embeddings (Gehring et al., 2017). Technically, instead of using sinusoidal functions to compute a static embedding, a trainable positional embedding $pe_i \in \mathbb{R}^d$ is randomly initialized for each position and is updated during training together with the model weights. This includes BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019) and their derivatives used in our thesis. One drawback with learned position embeddings is that such a pre-trained model cannot process a sequence that is longer than the model's pre-defined maximum length.

2.2.3 Transformer layers

In addition to multi-head attention, each Transformer layer uses some extra modules such as a feed-forward layer, layer normalization (Ba et al., 2016) and residual connec-

tions (He et al., 2016). The feed-forward layer FFN() with a feed-forward size d_{ffn} used in the Transformer is made up of two trainable weight matrices $W_1 \in \mathbb{R}^{d \times d_{ffn}}$ and $W_2 \in \mathbb{R}^{d_{ffn} \times d}$ and bias terms $b_1 \in \mathbb{R}^{d_{ffn}}$ and $b_2 \in \mathbb{R}^d$. For each input $x \in \mathbb{R}^d$ in a sequence of input $X \in \mathbb{R}^{L \times d}$, the feed-forward module computes

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

where the max() function corresponds to the rectified linear unit (Fukushima, 1975) used by Transformer. Usually, the feed-forward size is larger than the model's hidden dimension $d_{ffn} > d$, so the module up-projects then down-projects its input. Since the input to the feed-forward layer in the Transformer model is a concatenation of all heads, the feed-forward module facilitates information exchange between heads.

A residual connection around a function sums up the output of the function and the input of the function. In other words, a residual connection around a function $F()$ with input a gives the outcome $F(a) + a$. It is useful in passing the original input information to the next layer when a layer fails to learn a meaningful transformation. Layer normalization LayerNorm() is a normalization technique that transforms the input to a layer by subtracting its mean and dividing by its standard deviation. It is applied to each input sequence independently regardless of how inputs are batched. These two techniques are understood to stabilize training and help with model generalization.

2.2.4 Transformer encoder

With input X_{l-1} from the previous layer, the output of a single Transformer encoder layer X_l can be computed as:

$$\begin{aligned} X_l &= \text{TransformerEncoderLayer}(X_{l-1}) \\ &= \text{LayerNorm}\left(X'_l + \text{FFN}(X'_l)\right) \\ \text{where } X'_l &= \text{LayerNorm}\left(X_{l-1} + \text{MultiHeadAtt}(X_{l-1}, X_{l-1})\right) \end{aligned}$$

The stacking of multiple attention layers is illustrated by the "$N\times$" notation in Figure 2.1. For a Transformer encoder with N_{enc} layers, the above operations are sequentially done for each layer $l \in \{1, 2, \ldots, N_{enc}\}$, with input X_0 taken as the input word embeddings described in Section 2.2.2. Note that each layer has its own trainable weights. The output of the final encoder layer $X_{N_{enc}}$ is used as the input to the decoder.

Hence we can represent the Encoder operation as:

$$\begin{aligned} X_{N_{enc}} &= \text{TransformerEncoder}(X_0) \\ &= \text{TransformerEncoderLayer}_{N_{enc}}(\\ &\quad \text{TransformerEncoderLayer}_{N_{enc}-1}(\\ &\quad \dots \\ &\quad \text{TransformerEncoderLayer}_1(X_0)\dots)) \end{aligned}$$

2.2.5 Transformer decoder

The decoder has stacked multi-head attention layers similar to the encoder, but there are some important differences. First, in the self-attention over target-side input tokens, the attention uses a mask matrix to prevent attention to future tokens as we detailed in Section 2.2.1. In other words, given a sequence of target-side inputs $Y = [y_1, y_2, \dots, y_M]$, for each input y_t at position (time step) t, the attention mechanism can only access $y_1, y_2, \dots$ until y_t itself. Next, each decoder layer has extra multi-head attention that attends to the outputs $X_{N_{enc}}$ from the final encoder layer. This is to allow the decoder to access the encoder outputs and make predictions based on both the source and target sides. Using Y_{l-1} as the input from the previous decoder layer, the output of a single Transformer decoder layer Y_l can be written as:

$$\begin{aligned} Y_l &= \text{TransformerDecoderLayer}(Y_{l-1}, X_{N_{enc}}) \\ &= \text{LayerNorm}\left(Y_l'' + \text{FFN}(Y_l'')\right) \\ \text{where } Y_l'' &= \text{LayerNorm}\left(Y_{l-1}' + \text{MultiHeadAtt}(Y_{l-1}', X_{N_{enc}})\right) \\ \text{and } Y_l' &= \text{LayerNorm}\left(Y_{l-1} + \text{MultiHeadAtt}(Y_{l-1}, Y_{l-1})\right) \end{aligned}$$

Similarly to the encoder layers, the decoder layers are also stacked and computed N_{dec} times sequentially. The first layer input Y_0 are the word embeddings of Y. The Decoder output can be expressed as:

$$\begin{aligned} Y_{N_{dec}} &= \text{TransformerDecoder}(X_0) \\ &= \text{TransformerDecoderLayer}_{N_{dec}}(\\ &\quad \text{TransformerDecoderLayer}_{N_{dec}-1}(\\ &\quad \dots \\ &\quad \text{TransformerDecoderLayer}_1(Y_0, X_{N_{enc}}), \dots, X_{N_{enc}}), X_{N_{enc}}) \end{aligned}$$

In the Transformer paper, it is set that the encoder and the decoder have the same number of layers $N_{enc} = N_{dec}$.

2.2.6 Decoder output layer

To perform sequence generation, the output from the last decoder layer $Y_{N_{dec}} \in \mathbb{R}^{M \times d}$ passes through a learnable linear transformation (often called an output layer or a language model head) $W_{out} \in \mathbb{R}^{d \times |V|}$ and then a softmax function to obtain a distribution over the entire vocabulary for each step t in the output sequence. If we let $[i]$ denote the operation of retrieving the i-th entry of a vector or a matrix, then the probability of generating the v-th token from vocabulary V at time step t is conventionally computed as:

$$P\left(y_t = V\left[v\right] | y_{<t}, X; \theta\right) = \text{softmax}\left(Y_{N_{dec}}\left[t\right] W_{out}\right)\left[v\right]$$

Similar to the attention softmax, the softmax operation here suppresses extremely large or small values to between 0 and 1 and offers an interpretation of probability distribution since the output elements add up to 1. The entire output from $\text{softmax}(Y_{N_{dec}} W_{out})$ can be treated as probability distributions over the whole vocabulary for all time steps. This vocabulary probability computation is used in both inference and training stages as we have explained in Section 2.1.

The output layer is shared across all time steps. It has also been a convention to use the same embedding matrix E_{word} for both the encoder and the decoder if the vocabulary is jointly learned on the source and target data. Usually, the output layer weights can also be tied to the embedding matrix $W_{out} = {E_{word}}^T$ for both quality and memory efficiency considerations especially in small Transformer models. This is commonly referred to as *embedding tying* (Press and Wolf, 2017), which we also actively use in our work.

2.3 Data Augmentation

2.3.1 Overview

Supervised learning aims to minimize a model's loss on a training set that contains many input-output pairs $D_{train} = \left\{(X_1, Y_1^*), (X_2, Y_2^*), \ldots, (X_{|D|}, Y_{|D|}^*)\right\}$. Yet, the ultimate goal is to deploy the model in situations where the model inputs do not exist in the training data. Such a model's performance is measured by how well it generalizes

to unseen test data D_{test} as explained previously. To achieve this, it usually requires a large amount of training data which remains unavailable in many cases—languages, domains, tasks, etc.

Since it is expensive to employ human annotators to create training resources, automatic data augmentation is one common way to alleviate this problem. It has received more attention recently as the NLP community expands to more domains and tasks as well as adopts larger models (Feng et al., 2021; Chen et al., 2023). Data augmentation aims to automatically form new training examples, thereby expanding the quantity and diversity of the training data. Practically, data augmentation creates training pairs (X', Y') different from the existing data but should improve the performance of a neural network.

There appears to be no single definitive way to categorize data augmentation methods among previous surveys. Feng et al. (2021)'s survey considered the means of generation (rule-based, example interpolation-based, and model-based) as well as the applicable downstream tasks. Somewhat similar, Li et al. (2022) used three categories: paraphrasing-based, noising-based, and sampling-based. Hedderich et al. (2021) classified different methods by resource requirements (e.g. labelled data, unlabelled data, manual heuristics) and outcomes (additional data, better representations, etc.). Another paper surveys related works by granularity, for example, character-level, token-level, sentence-level, or hidden space-level augmentation (Şahin, 2022; Chen et al., 2023)

In the following sections, we briefly review data augmentation techniques relevant to our thesis and introduce them under three categories established by us. We discuss the benefits and limitations of different methods and draw links between them and the techniques we use later. It is worth noting that these categories are appropriate for our work, but they are neither comprehensive in the NLP literature nor mutually exclusive. We introduce them based on how the extra data is formed and what types of additional materials the augmentation process depends on.

2.3.2 Self-data augmentation

We define self-data augmentation as only utilizing the existing training data to augment model training with few external resources: the reliance on other tools or data in any form remains minimal. A popular line of work is to add noise to the genuine data to form new data pairs (Wei and Zou, 2019). It can be done in various ways to alter the input or the output, where most of the previous papers noise the input side but maintain

the integrity of the output label. Namely, stemming from a valid training example (X, Y^*), the method creates new input-output data (X', Y^*) to provide better coverage of the possible input space to improve the model's robustness to input variations and noise. This line of approaches is easy to implement and does not require complex tools or additional data.

Straightforward and universal methods include token-level noising, deletion, insertion, and swapping (Xie et al., 2017; Wei and Zou, 2019), which has been applied to both text classification and generation. Specifically for machine translation, a simple method from Currey et al. (2017) copies the output over to the input end, where the synthesis method can be seen as an identical mapping from the target side to the source side. This thesis adopts these methods for word modelling, definition generation, and code generation.

2.3.3 Retrieval-based augmentation

Under this category, we regard both training inputs X and gold outputs Y^* as existing natural texts which can form a perfect pair of training data (X, Y^*), but they remain separated in the wild and thus unavailable for supervised learning. Retrieval-based augmentation aims to automatically construct complete data pairs by retrieving the inputs X and outputs Y^* scattered over a large volume of data. The augmented data samples, if accurately retrieved, will be of high quality as the inputs and outputs are initially both human-created. Another advantage is that the method can be carried out on a very large scale since raw data are vastly available on the web.

Retrieval-based methods do not create new instances, but rather, automatically find and construct data that can suit the learning of a specific task. Related works involve automatic data creation for text translation and summarization at scale (Resnik, 1999; Rush et al., 2015). The former is explored in our thesis. It aims to find sentences in different languages that have the same meaning from a large collection of data and pair them up to form data for training translation systems. Major efforts concentrate on retrieving from large public resources like the web (Resnik and Smith, 2003). Whilst some pipelines need to align web pages using their domains as a heuristic before comparing sentence similarity, our work in Chapter 3 proposes a parallel sentence aligner without such a requirement.

2.3.4 Generation-based augmentation

Finally, we consider methods that use models or sophisticated rules to generate new training pairs. The distinction from the aforementioned categories is that these works use some supervised mechanism to create new training samples. The method is supervised in the sense that the data creation relies on models or rules that are inferred from existing gold data or knowledge. Many methods belonging to this category come with the advantage that the seed data only needs to exist on one side (either the input or the output), which is often the case in NLP where abundant unlabelled text can be found without the structure of a complete data pair. The disadvantage is that the newly generated data is usually of lower quality.

Previous research has explored token-level replacement, focusing on substituting words in the input with similar words, paradigmatic words, or rare words while preserving the validity of the output. This usually needs a set of pre-defined rules, a dictionary, or a language model (Zhang et al., 2015; Fadaee et al., 2017; Chen et al., 2020a). A higher-level consideration is to paraphrase the input side while keeping the output unchanged (Kumar et al., 2019). In machine translation, early attempts include using synthetic parallel data for statistical models (Bojar et al., 2013; Tamchyna and Bojar, 2015). A commonly adopted technique in the neural era is back-translation, which creates pseudo-parallel data by machine-translating monolingual target-side data into the source side and pairing the translated source with the original target (Sennrich et al., 2016a). Similarly, it is also possible to create the target-side data from the source (Zhang and Zong, 2016). These techniques have especially inspired our data augmentation methods for generation tasks involving programming languages.

2.3.5 Benefits and risks

From a high-level consideration, data augmentation equates to using more (but probably lower-quality) training data to achieve better generalization. This also helps a model to be less over-parameterized relative to the training sample size. We provide two further interpretations of the benefits that can be brought by data augmentation, specifically from the perspective of data variations and external information. The first is the regularization effects. Both valid variations and noise signals in the augmented data expose the model to wider scenarios that it may encounter at the test time. More varied and noised training data can make the original task optimization more challenging, but it also leads to a more robust trained model (Cheng et al., 2020; Li and Specia,

2019). From a multi-tasking point of view, a model trained on augmented data with auxiliary tasks can implicitly derive better internal representations (Wei et al., 2021) or explicitly learns to put more weight on the appropriate input information (Sánchez-Cartagena et al., 2021).

Data augmentation can also be thought of as a way to fuse in external prior information that is not present in the original training data. From dictionary-based word replacement to data synthesis using auxiliary models or hand-crafted rules, the information in those extra tools or human knowledge is distilled into the augmented data and subsequently transferred to the models trained on the augmented data. For example, both back-translation (Sennrich et al., 2016a) and target copying (Currey et al., 2017) introduce genuine target-side text that can be orders of magnitude larger than the parallel data, which directly leads to a stronger decoder language model.

On the other hand, we note that data augmentation can have adverse effects if not conducted carefully. The most detrimental could be the introduction of erroneous data, where the input-output correspondence is no longer maintained (Wei and Zou, 2019). Consequently, a neural network trained on such data may learn to produce incorrect or non-optimal output for an input. Next, the addition of out-of-domain or irrelevant data can lead to a domain shift in the trained model, which poses a challenge if a model is designed to deal with domain-specific texts or tasks. Finally, augmentation techniques that rely on existing data as a seed could result in bias amplification, where the bias in the existing data propagates into the augmented data and eventually becomes more prevalent. It is important for practitioners to evaluate and steer away from these risks before selecting the data sources and tools for data augmentation.

Chapter 3

Generative Retrieval of Parallel Data

3.1 Introduction

Having large and high-quality data is important for neural machine translation (NMT), which needs data in the form of parallel sentences. The language of the input sentence is often called *source* and the output side in the desired language is called *target*. Employing human annotators to create such source-target pairs is extremely expensive, so a common way is to extract sentences with the same meaning from corpora containing text in different languages. One way to construct such corpora is to mine the web and use the crawled content to form large collections indexed by language (Resnik and Smith, 2003). To locate parallel sentences across two languages, a naive way is to measure sentence similarity between all possible sentence pairs and extract the top-scoring ones. This poses two major challenges:

1. Accurately determining the semantic similarity (parallelism) of a sentence pair across two languages.

2. Efficiently scoring sentence similarity for all possible pairs from two languages in order to find true parallel sentences.

Scoring each sentence in one language against each sentence in another language results in unaffordable quadratic time complexity.

A typical workflow reduces the search complexity in a coarse-to-fine manner by aligning documents (web pages), with simple heuristics such as the domain or the page structure, and then aligning sentences across the document pair (Uszkoreit et al., 2010). However, websites containing parallel sentences may not have matching document

structures, and this approach limits the mining process to only web pages intended to be parallel in the same domain.

Some recent works investigated direct sentence alignment from two monolingual corpora without the need for document-level alignment or even document structure. The results from the Building and Using Comparable Corpora (BUCC) shared tasks show that direct sentence alignment can be done by comparing sentence-level lexical representations, neural representations, or a combination of the two (Zweigenbaum et al., 2017, 2018). Another line of research maps all sentences to multilingual sentence embeddings. It compares them using a modified cosine similarity metric by applying k-nearest neighbours (k-NN) as a constraint on the search space (Artetxe and Schwenk, 2019a).

In this chapter, we propose a novel solution which uses an NMT system to generate and retrieve potentially parallel sentences from a pile of data. In terms of similarity scoring, we use an NMT system to force-decode possible sentences in the target language given a source sentence and treat the model's probability scores as a similarity measure for candidate source-target sentence pairs. The way we avoid pairwise scoring is inspired by constrained decoding in NMT, where the choice of output tokens at each time step is constrained to a pre-defined list (Hokamp and Liu, 2017). It works as follows: We build a trie of all target sentences. Then we translate each source sentence to the target language but constrain left-to-right beam search to follow the trie. In other words, at any time, every translation hypothesis is a prefix of some sentence(s) in the target language. Rather than freely choosing which tokens to extend by, a hypothesis is limited to extensions that exist in the target language corpus. In effect, at each time step, constrained beam search limits the target language candidates for each source sentence.

Our work can be deployed with a seed translation system to continuously augment the training data, or it can be used as a standalone corpus miner. This chapter brings two insights corresponding to the challenges in parallel sentence retrieval we identified earlier:

1. Similarity: instead of comparing translated text or neural embeddings, we use a machine translation model to directly score and retrieve sentences on the fly.

2. Efficiency: trie-constrained beam search, where only the top-scoring prefixes need to be considered at each decoding step, is used to approximate pairwise comparison of full sentences.

We also point out an imperfection with a popular evaluation benchmark. Finally, our code implementation is publicly available.[3]

3.2 Related Work

3.2.1 Document alignment and sentence alignment

A typical parallel corpus retrieval workflow first aligns documents in two different languages, in order to limit the search space for subsequent sentence alignment procedures. Early document alignment methods utilized the webpage structure (Resnik and Smith, 2003; Shi et al., 2006). Later, Uszkoreit et al. (2010) proposed to translate all documents into a single language (usually English), and shortlist candidate document pairs based on TF-IDF-weighted n-grams. A large parallel corpora release named ParaCrawl adopted this approach (Bañón et al., 2020). Moving from string features to neural features, Guo et al. (2019) suggested comparing document embeddings obtained from neural sentence embeddings.

Within an aligned document pair, sentence alignment can be done by comparing sentence length in words (Brown et al., 1991) or characters (Gale and Church, 1993), which is then improved by adding lexical features (Varga et al., 2005). After translating texts into the same language, string-based metrics like BLEU can be used to determine parallel texts, by anchoring the most reliable alignments first (Sennrich and Volk, 2011). Most recently, Thompson and Koehn (2019) proposed to compare bilingual sentence embeddings with dynamic programming in linear runtime. However, the assumption of document-level alignment is not always ideal because this limits the sentence alignment search space to intended parallel documents.

3.2.2 Direct sentence alignment

There are also research efforts on parallel sentence extraction without the reliance on document-level information, however, with the disadvantage that the comparison often requires quadratic time. Munteanu and Marcu (2002) acquire parallel phrases from comparable corpora using bilingual tries and seed translation dictionaries. Leong et al. (2018) used an autoencoder and a maximum entropy classifier. Bouamor and Sajjad (2018) considered cosine similarity between averaged multilingual word em-

[3] https://github.com/marian-nmt/marian-dev/tree/trieme

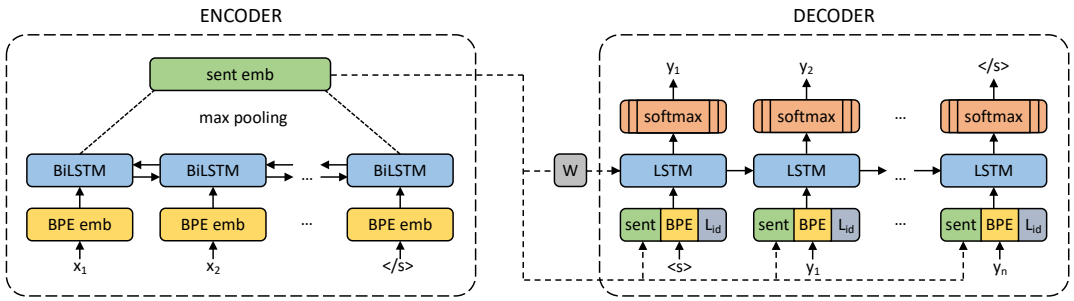


Figure 3.1: The LASER model illustrated by Artetxe and Schwenk (2019a).

beddings with downstream filtering. España-Bonet et al. (2017) has studied the use of translation-derived multilingual sentence embeddings to identify parallel sentences. Guo et al. (2018) designed a dual encoder model to learn multilingual sentence embeddings directly with added negative examples.

Our proposed method is compared with three distinct methods. The first work, named STACC, compares the Jaccard similarity of overlap between the word translations of a source with a target sentence, as well as the word translations of a target sentence with a source sentence (Etchegoyhen and Azpeitia, 2016). $\text{STACC}_{\text{weight}}$ extends this work by weighting frequent words less, which makes function words contribute less to the final similarity score (Azpeitia et al., 2017). Another enhanced version, $\text{STACC}_{\text{weight-penalty}}$, adds a penalty to named entity mismatches (Azpeitia et al., 2018).

The second model, named LASER, derives multilingual sentence embeddings from the hidden states in multilingual neural machine translation using an LSTM encoder-decoder architecture (Schwenk and Douze, 2017; Schwenk, 2018). As illustrated in Figure 3.1, all source word information flows through a hidden representation bottleneck, resulting from max-pooling word-level encoder outputs. The decoder generates a translation in the desired language using the representation, previous word, and a language indicator. This model is trained on different language pairs, and since the target language is known only at inference time, the bottleneck has to capture language-agnostic semantics in the source. The representation is used as the sentence embedding, and the decoder is discarded after training. Cosine similarity between sentence embeddings is used as a similarity measure. Corroborated by Guo et al. (2018), Artetxe and Schwenk (2019a) argued that the scale of cosine similarity is not globally consistent, so they improved sentence comparison by leveraging a margin-based score function which considers the candidate pair and the other nearest pairs. Later efforts scaled the embeddings to more languages (Artetxe and Schwenk, 2019b) and used

these to yield a large parallel corpus named WikiMatrix from Wikipedia (Schwenk et al., 2021).

Finally, we include an encoder-only sentence embedding work for comparison. Wieting et al. (2019) trained a multilingual LSTM encoder with a cosine objective to distinguish positive and negative translations given a source sentence. A sentence embedding is obtained by averaging sub-word embeddings, and they claim orders of magnitude encoding speed-up compared to Schwenk (2018).

3.2.3 Forced scoring and decoding

A neural machine translation model produces a beam-ful of candidate hypotheses with probability scores. The final output is determined by picking the one with the highest score. From this perspective, our work can be seen as relevant to cross-entropy scoring for parallel sentence filtering (Junczys-Dowmunt, 2018), which won the WMT 2018 shared task on parallel corpus filtering (Koehn et al., 2018). Both works use the intrinsic model scores as a source-target similarity indicator, yet a difference is that our work only uses the score from a single direction as opposed to dual-way cross-entropy scoring. However, Junczys-Dowmunt (2018)'s work is not directly applicable in parallel data retrieval because it operates at the sentence pair level to remove noisy pairs, whilst our search space is exponentially larger given that we need to compare each possible source-target sentence pair.

As an innovation, we speed up the comparison using constrained decoding (Hokamp and Liu, 2017; Post and Vilar, 2018), which has also been applied to image captioning (Anderson et al., 2017) and keyword generation (Lian et al., 2019) before our exploration. Constrained decoding allows us to approximate a global similarity comparison with consecutive unidirectional step-wise comparisons. We refer readers to the explanations in the next section.

3.3 Methodology

3.3.1 A translator can be a similarity scorer

Following the general sequence-to-sequence modelling in Section 2.1.1, neural translation systems can assign a negative log-likelihood to an arbitrary sentence pair by inputting a source sentence and performing forced decoding of a certain target sen-

tence. In other words, this can be viewed as computing a "similarity score" for a target sentence given a source input. Intuitively, we could score every possible pair of source and target sentences using a translation system. Then the pairs that score highly can be returned for further processing. Nevertheless, this strategy, despite being straightforward, will not scale given its quadratic time complexity. Hence, in Section 3.3.2, we introduce a way to approximate pairwise comparison with beam search constrained to a prefix tree, which is constructed using the corpus in the target language.

3.3.2 Trie-constrained decoding

A trie, also called a prefix tree, is a data structure that can compactly store natural language data by collapsing common prefixes together. As illustrated in Figure 3.2, "I like strudels" and "I like cakes" can share the same prefix "I like"; ultimately, all of the natural sentences share the same begin-of-sentence node. We extend the aforementioned source-target scoring method with a trie constraint that contains all sentences in the target language corpus. An autoregressive translation system generates translations one token at a time (typically in the same direction as how the target language is read by a person), so it can follow the trie of target language sentences by starting from the begin-of-sentence node and expanding into leaf nodes.

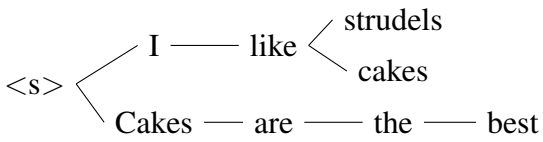


Figure 3.2: An illustration of a trie storing three sentences.

Formally, the typical translation process uses beam search to approximately maximise the probability of a target generation given a source input. We modify beam search to restrict all partial hypotheses in the beam to be a prefix in the trie constraint. In other words, a hypothesis needs to be the prefix of at least one sentence in the target corpus. In this way, when the translation is finished for any source sentence, we are guaranteed to match a sentence in the target corpus, as opposed to a freely generated one. We regard this sentence as a retrieved sentence that is parallel to the input according to the model.

A trie is an efficient data structure with which this prefix constraint can be assessed; partial translations are augmented with a pointer to track their position on the trie. We

consider two places to enforce the constraint:

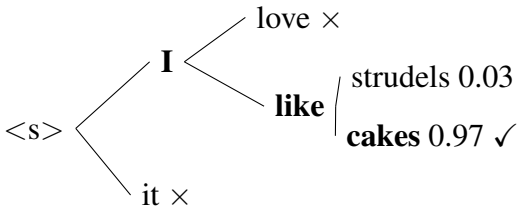
1. *Pre-expansion pruning*: At each time step, an NMT model generates a probability distribution over all tokens, but each hypothesis can only be expanded with the tokens corresponding to the children of its current trie node. This search process is guaranteed to find at least one target sentence for each source sentence. Downstream filtering is needed to remove false positives.

2. *Post-expansion pruning*: At each time step, beam search creates hypotheses for the next word with the full vocabulary. We then prune hypotheses to fit in the beam and further prune any hypothesis that is not a prefix of any target language sentence. The concept here is to "translate then prune". In practice, most candidate source sentences do not have a corresponding translation in the target corpus, so beam search can terminate early once all hypotheses are pruned.

Algorithm 2 Trie-constrained beam search: maximum output length L, beam size B, vocabulary V, and a pre-built $trie$ of target language sentences

$beam_0 \leftarrow \{<\text{s}>\}$
$completed \leftarrow \{\}$
for time step t in 1 to L **do**
  $beam_t \leftarrow \{\}$
  for hypothesis h in $beam_{t-1}$ **do**
    $V_t \leftarrow V$
    if pre-expansion **then** ▷ pre-expansion pruning
      $V_t \leftarrow V_t \cap \text{Children}(trie, h)$ ▷ pre-expansion pruning
    $beam_t \leftarrow beam_t \cup \text{Continue}(h, V_t)$
  $beam_t \leftarrow \text{NBest}(beam_t, B - |completed|)$
  if post-expansion **then** ▷ post-expansion pruning
    $beam_t \leftarrow beam_t \cap trie$ ▷ post-expansion pruning
  Move hypotheses ending with </s> from $beam_t$ to $completed$.
  if $beam_t$ is empty **then**
    return Best($completed$)
return Best($completed$)

Pseudo-code in Algorithm 2 outlines both variants of our constrained beam search algorithm. In addition to canonical beam search introduced in Section 2.1.2, we use "▷"



- *Source sentence*: Me gustan los pasteles (I like cakes)
- *Target trie constraint*: as shown in Figure 3.2

Figure 3.3: An illustration of trie-constrained decoding with post-expansion pruning and beam size 2. × denotes a pruned hypothesis that is not in the trie, a number denotes the translation probability of a completed hypothesis, and ✓ denotes the retrieved sentence.

symbols to highlight the position where post-expansion pruning or pre-expansion pruning takes place. For pre-expansion pruning, the full vocabulary needs to be trimmed at each time step; then the hypotheses in the beam are only expanded with the trimmed vocabulary. On the other hand, for post-expansion pruning, the hypotheses are expanded first, but the beam can only keep the ones that intersect with the pre-built trie. In other words, a hypothesis can only remain in the beam if it is a prefix in the trie (otherwise there is no chance to match a complete sentence in the target-side corpus). We visualize in Figure 3.3 trie-constrained beam search with post-expansion pruning to aid understanding.

Running naive forced scoring (decoding) between N source language sentences and M target language sentences would end up with a time complexity of $O(MN)$. The modified beam search algorithm allows us to efficiently approximate the comparison between a single source sentence and all M target sentences step by step at inference time. We let B denote the beam size and L denote the maximum output length. Given each source sentence, our NMT decoder only expands the top B hypotheses intersecting with the trie, for at most L times, regardless of M. In comparison, our proposed method can reduce the naive NMT forced scoring from $O(MN)$ to $O(BLN)$ in complexity, where the product of the beam size and the maximum length is much smaller than the target corpus size, $BL \ll M$.

3.3.3 Sentence pair filtering

Pre-expansion pruning leaves each source sentence with a candidate in the target language, which ought to be filtered out if not deemed parallel. We propose to try two methods. First, when NMT generates an output sentence for an input, a negative log probability can be computed as a by-product score. One way to perform filtering is to only keep sentence pairs with a per-word score higher than a certain threshold. The advantage of this approach is that it does not require additional computation in our case, since constrained decoding produces this score intrinsically. Another choice can be using an off-the-shelf tool that scores sentence similarity at the sentence-pair level to not add time complexity. In our work, we select Bicleaner (Sánchez-Cartagena et al., 2018).[4] Such filtering is optional for post-expansion pruning.

3.3.4 Efficient implementation

The trie used in our NMT decoding should be fast to query and small enough to fit in memory. We use an array of nodes as the basic data structure. Each node contains a key corresponding to a vocabulary item, as well as a pointer to another array containing all possible continuations at the next depth level. Binary search is used to find the correct continuations to the next level. Thanks to byte-pair encoding (BPE, Sennrich et al., 2016b), theoretically, the maximum vocabulary size can always be limited to below 65536. This allows the use of 2-byte integers as keys to minimise memory usage.

To integrate the trie into an NMT decoder, we maintain external pointers to possible children nodes in the trie for each active hypothesis in the beam during inference. When the hypotheses are expanded at each time step, the pointers are also advanced to the next trie depth level. This ensures that cross-referencing to evaluate the trie constraint has a negligible effect on the original decoding speed.

3.4 Experiments

3.4.1 The BUCC shared task

We evaluate our method on the BUCC shared task, which asks participants to align parallel sentences across large monolingual datasets of English and other languages

[4] https://github.com/bitextor/bicleaner

(Zweigenbaum et al., 2017, 2018). In those corpora, monolingual and originally parallel sentences are sourced from Wikipedia and News Commentary respectively, and are mixed to create artificial data pools in separate languages. Around 2% of the sentences are parallel depending on the language pair. English is always one candidate language, and in our experiments, we align sentences from the French (fr), German (de), and Russian (ru) corpora with English. This will give parallel sentences in English-French, English-German, and English-Russian.

For each language (pair), the data are divided into sample, training, and test sets at a ratio of 1:10:10. The sample and training sentence alignments are public, but the gold alignments for the test set are kept private. To benchmark system performance on the test set, participants will need to contact the task organizers for evaluation. The evaluation metrics adopted in this task are precision P, recall R, and $F1$ score with precision and recall weighted equally:

$$F1 = \frac{2 \times P \times R}{P + R}$$

Precision measures the proportion of the retrieved sentence pairs that are correct, while recall measures the proportion of gold sentence pairs that are retrieved. For downstream machine translation training, precision would imply the quality of the parallel data and recall would imply the quantity of the parallel data. Evaluating systems using F1 scores prompts a balance between the two factors.

3.4.2 Train-test overlap in BUCC

Our manual inspection of the BUCC shared task data revealed that there are overlapping parallel sentences across the sample, train, and test sets.[5] For example, more than 60% of the German-English gold alignments in the test set appear in the training set too. It leads to an unfair evaluation if a system memorizes the training split alignments and is then compared with other systems in the evaluation. We believe that this is a meaningful contribution to our field, given that the BUCC shared task has been held twice and the benchmark has been then used in many papers investigating parallel sentence mining.

The shared task organisers confirmed the issue some time after we pointed it out to them. They re-evaluated previous submissions' test performance on the benchmark

[5]The inspection was conducted in a post hoc manner after we discovered that fine-tuning our translation models on the training split until overfitting did not degrade performance on the sample split—at that time we had already performed some experiments. We re-ran some of the experiments to make sure our translation models do not memorize the gold alignments.

without considering the overlapping parallel sentences. According to the organisers' reply, on average, recall drops by 2% with the largest being 4% among all submissions, but they did not disclose the change in precision and we do not have access to the updated test performance for individual systems. As a preventive measure against overfitting, in our work, we do not train our NMT systems on the BUCC training data.

3.4.3 Translation model configurations

We trained separate models to translate each language of interest (French, German, Russian) into English. This is an efficiency consideration that, since the mining process is English-centric, the English trie only needs to be built once, and it can be re-used regardless of the source language. All models use a Transformer architecture with 6 layers, 512 hidden size, 2048 feed-forward size, and 8 heads. They are trained using the Marian NMT framework (Junczys-Dowmunt et al., 2018a). We used parallel data from the WMT15 news translation task (Bojar et al., 2015), excluding the News Commentary data as required by BUCC. It also prevents our systems from memorizing the gold parallel sentences, especially given the overlap issue. We also did not optimize training hyperparameters towards the BUCC training set alignments. Table 3.1 details the data composition for training each language pair. We processed the training data by applying BPE with a joint vocabulary sized at 32K for each language pair.

	en–fr	en–ru	en–de
Europarl	✓		✓
Common Crawl	✓	✓	✓
UN corpus	✓	✓	
Yandex		✓	

Table 3.1: WMT15 data resources used for our models in BUCC.

After all translation systems were trained, we decided on a beam size of 90 for constrained decoding by performing a grid search on the German-English sample set without changing it for further experiments. This can be interpreted as tracking the top 90 partial hypotheses (prefixes) based on model probability scores at each inference step. While this is an unusually large beam size in translation decoding, we found it to be optimal for parallel sentence retrieval on the sample set.

Regarding the filtering setup for pre-expansion pruning, per-word score thresholds

are tuned separately for each language pair, because languages inherently have different entropy values. For Bicleaner, we stick to its default settings, except that we disable the language model filter since translation cross-entropy is indicative of the target language model scores itself. Through filtering, our objective is to maximize the F1 score by adjusting the trade-off between recall and precision.

3.4.4 Pruning, filtering, and translation direction

Pruning and filtering We initially used the BUCC sample data as our internal test. Table 3.2 reports the performance of different system settings on the sample split for French-English, Russian-English, and German-English. We see that pre-expansion pruning is a better choice for applying the trie constraint as it exceeds post-expansion pruning by around 10 F1 points for all language pairs. This could be explained by the fact that the decoder has a better chance of generating the correct target sentence if the available vocabulary is constrained, preventing high-scoring tokens that do not fulfil the constraint from occupying the beam. Cross-entropy filtering and Bicleaner filtering both are effective in removing non-parallel sentences, leading to a preference towards precision for all languages, which is especially the case for per-word cross-entropy filtering. Hence, we propose to take a combination of candidates from the two filtering methods, where better F1 scores are achieved with more balanced precision and recall.

	fr→en			ru→en			de→en		
	P	R	F1	P	R	F1	P	R	F1
post-expansion pruning	92	62	74	**99**	61	75	88	61	72
pre-expansion pruning									
+ cross-entropy (CE)	**97**	72	83	98	**84**	**90**	**96**	73	83
+ Bicleaner (BC)	86	77	81		n/a†		93	81	86
+ CE ∪ BC	93	**81**	**86**		n/a†		91	**84**	**87**

†Bicleaner does not have a published classifier for Russian-English.

Table 3.2: Precision, recall, and F1 scores on the BUCC sample set for different pruning and filtering choices.

Translation direction All our initial models translate into English whereby the input is non-English and the trie constraint is the English corpus, but our method should be

language-agnostic in theory. Also, as our method still exhibits a much higher precision than recall, we hypothesise that a system trained in the inverse direction might retrieve different sentence pairs, and then taking a union of both directions will sacrifice some precision for recall and consequently a higher F1 score. We thus run an additional experiment for German-English alignment using an English-to-German translation system instead. It can be used to mine English-German sentences with the same constrained decoding, but with English as the source input and German sentences put on a trie constraint. This data-controlled setting can help to test whether translation direction will affect the parallel sentence mining performance and whether it is beneficial to combine sentences retrieved by translation models in opposite directions.

	de→en			**de←en**			→ ∪ ←		
	P	R	F1	P	R	F1	P	R	F1
post-expansion	88	61	72	96	59	73	81	75	81
pre-expansion									
+ cross-entropy (CE)	96	73	83	**98**	79	88	96	87	**91**
+ Bicleaner (BC)	93	81	86	91	82	86	86	87	87
+ CE ∪ BC	91	84	87	91	90	86	88	**91**	**91**

Table 3.3: Impact of the translation direction on the BUCC German-English sample set performance.

We present our investigations using `de→en`, `en→de`, and their union in Table 3.3. It is seen that the two unidirectional models have similar precision, recall and F1 scores. By considering the union of the retrieved parallel sentences, we find that, as expected, precision is sacrificed but both recall and F1 scores improve. Consistent with our previous observation, pre-expansion pruning still outperforms post-expansion vocabulary pruning.

3.4.5 Comparisons on the BUCC benchmark

In order to directly compare with previous works, we stick to the best pre-expansion variant found on the sample set and apply it to the German-English training and test splits. We run the model on the training set to decide on a filtering threshold that maximizes the F1 score and apply the same setting to the test split. This similar thresh-

System	Train F1	Test F1
$\text{STACC}_{\text{weight}}$ (Azpeitia et al., 2017)	83.3	83.7
$\text{STACC}_{\text{weight-penalty}}$ (Azpeitia et al., 2018)	84.3	85.5
LASER_{18} (Schwenk, 2018)	76.1	76.9
LASER_{19a} (Artetxe and Schwenk, 2019a)	94.8	95.6
LASER_{19b} (Artetxe and Schwenk, 2019b)	95.4	96.2
encoder-SP_{20K} (Wieting et al., 2019)	76.9†	n/a
encoder-SP_{40K} (Wieting et al., 2019)	77.5†	n/a
ours (pre-expansion + CE ∪ BC)	83.0	83.9
+ in-domain tuning	85.5‡	n/a

†Evaluated on the training set in a zero-shot fashion.

‡Fine-tuned on News Commentary with BUCC training set alignments removed.

Table 3.4: F1 scores of ours and other methods on BUCC German-English training and test sets.

old tuning has also been performed by the variants of STACC (Azpeitia et al., 2017, 2018) and LASER (Schwenk, 2018; Artetxe and Schwenk, 2019a,b) for the purpose of maximizing F1. The systems from Wieting et al. (2019), denoted as encoder-SP, were benchmarked on the training set in a zero-shot fashion. Notation-wise, we use LASER_{yy} to denote the checkpoint published in year yy, and we use encoder-$\text{SP}_{|V|}$ for Wieting et al. (2019)'s model with vocabulary size $|V|$.

We report training and test F1 scores in Table 3.4 together with other models; the test set results have been computed by BUCC organizers. It is worth noting that the result numbers of other systems in this table are taken from their respective published papers without adjusting for the performance drop after re-evaluation as discussed in Section 3.4.2. We list past works' variants that attained the best F1.

At the bottom of Table 3.4, we use an extra experiment to understand how our system performs with in-domain data. We fine-tune our `de`→`en` and `en`→`de` translation systems on News Commentary, but only after we manually exclude the sentence pairs where either the German or the English side appears in BUCC training or test splits. This way, we prevent the model from directly memorizing the gold alignments. We note that BUCC submissions are asked not to use News Commentary at all, so comparison with previous works is only limited to the training set as an indicator.

3.5 Results and Discussions

3.5.1 Result comparison

Table 3.4 shows that our method achieves substantially higher numbers compared to the encoder-SP architecture, and at least comparable scores to STACC, taking into account the performance drop after the official re-evaluation due to the train-test overlap. Our numbers are favourable when compared to the early version of LASER, but underperforms when the margin-based cosine similarity metric is in place and the architecture is scaled to more layers and more languages. We attribute the later LASER variants' success to the adoption of a margin-based cosine criterion without the assumption of an orthogonal basis of the embedding space, which is not guaranteed in neural embeddings.

We need to note that these comparisons are not strictly fair since different works used different model architectures or training data. Our models use the vanilla Transformer-base architecture trained on WMT 2015 data, which are nowhere close to state-of-the-art NMT systems. We discover that our translation system sometimes yields a fluent sentence in the trie rather than a translation ("hallucination"). This could be attributed to the problem of domain mismatch (Koehn and Knowles, 2017) since we did not use any in-domain BUCC training data. Accordingly, we observe a gain in F1 after our systems are fine-tuned on in-domain data, as shown in the last row in Table 3.4.

3.5.2 Comparison pool size

By looking at Table 3.2 and Table 3.4, we observe an F1 score drop in our method when moving from the small sample set to the larger training set. When the candidate sentence pool expands, for each source sentence there are more possible targets to consider and therefore more tokens to consider at each step. We suspect that this could lead to a worsened performance due to having a higher chance of pruning incorrectly. We notice the same trend in other BUCC submissions which report their scores on both the sample and test splits (Azpeitia et al., 2018; Leong et al., 2018). Since the BUCC training is far from the "web size", this implies these methods might not scale well. The state-of-the-art LASER might deal with this better, as the margin-based scoring only considers a fixed candidate size locally. We observe a performance leap when the margin-based criterion (Artetxe and Schwenk, 2019a) replaced global cosine

comparison (Schwenk, 2018).

3.5.3 Limitation due to directionality

We argue that one problem of our method is associated with the inherent limitation of beam search decoding—it can be trapped in local optima. At a higher level, a genuine parallel target sentence cannot be recovered once it is pruned. Illustratively, "Por el momento, estoy bebiendo un café" (English: "At the moment, I am drinking coffee") can hardly match "I am drinking coffee at the moment" because a translation system will have a low chance of keeping a reordered translation prefix in its beam.

A larger beam size can ameliorate this problem but it also raises the risk of grabbing a high-scoring hallucination. As the generation time step gets larger, the chances of choosing an incorrect path will aggregate. Hence, we consider this generative retrieval approach better suited for domains with shorter and more deterministic phrases, such as matching queries and keywords (Lian et al., 2019). Whilst we opted for generative retrieval which can only be carried out in a single direction with autoregressive decoding, LASER operates on neural sentence embeddings, which might have a "holistic" view of sentences, and consequently are less sensitive to re-ordering and are not associated with the problem of being trapped in local optimal.

3.5.4 Test set imperfection

Finally, we discuss the limitations of the evaluation benchmark in addition to the train-test overlap explained earlier. We notice that some parallel sentence pairs in BUCC data are not included in the gold alignments. For instance, in the De-En training set, "de-000081259" and "de-000081260" are the same German sentence, and so are "en-000036940" and "en-000036941" on the English side. Gold alignments only include (de-000081259, en-000036940) and (de-000081260, en-000036941), but not the other two combinations.

3.6 Summary

This chapter presented a parallel data augmentation method by translating a source sentence with a target trie as a constraint. We proposed two constraint variants: at each inference step, we can prune against the trie before expanding hypotheses or after. Experiments showed that the former is more promising. When benchmarked

on the BUCC parallel sentence mining task, the system achieved scores comparable to several other works except for the later versions of LASER. We attribute the major limitation of our work to pruning error accumulation when the target language data pool is large and sequential generation errors. We also inspected the current evaluation benchmark and pointed out two defects.

We anticipate our method to have high extensibility to more sequential data retrieval tasks because technically the constrained decoding scheme can be used with an arbitrary autoregressive model learned on any sequence generation task. To avoid the limitations identified in this chapter, the algorithm could be used in domains where the output is more deterministic and concise such as structured data retrieval. Nonetheless, a generic drawback is that the method requires a seed system to begin with. It is realistic for languages with at least moderate resources to build a reasonable generation system, but gathering data for extremely low-resource tasks can be difficult.

Chapter 4

Data Enhancement for Unified Word and Definition Generation

4.1 Introduction

A monolingual dictionary is a large-scale collection of words paired with their definitions, for example, "retrieval" and "the process of finding and bringing back something" as listed in the Cambridge Dictionary.[6] Conventional dictionaries are indexed by words, usually sorted alphabetically, and display the possible meanings of these words. Although dictionaries are valuable educational resources, there are still certain aspects that can be improved upon. First, although a traditional dictionary can store a comprehensive list of definitions for each word entry, it can neither determine the specific meaning of a word in a context like "bank", nor define an unseen word like "influencer" until its definition is added. Also, the task in the inverse direction—finding a word to describe a concept—is infeasible.

Recent advancements in deep learning enabled researchers to build computer programs backed by neural networks to alleviate these issues (Hill et al., 2016b; Noraset et al., 2017). To define a word, a neural encoder can produce a contextualized representation of a word given its surrounding words, based on which a decoder can be trained to generate a definition of the word in the context. Neural models might also have the emergent capability to produce reasonable definitions for new and compound words by leveraging sub-word tokenization. On the other hand, to find a word for a phrase, a neural network can be used to encode the phrase and produce a fixed-sized vector representation to compare with the representations of all candidate words; the

[6] https://dictionary.cambridge.org/dictionary/english/retrieval

candidate with the closest representation to the phrase is selected as the answer.

Formally, in natural language processing, the task of producing a textual definition for a word is called *definition modelling* or *definition generation*; the inverse task of retrieving a word given a definition is referred to as *reverse dictionary*. Lately, the two tasks are approached using neural networks, and in turn, they help researchers better understand word sense and embeddings (Bosc and Vincent, 2018; Mickus et al., 2022). Research in this direction can further benefit users of low-resource languages when high-quality dictionaries are not available (Yan et al., 2020). Neural models capable of performing definition modelling can be a standalone neural dictionary that is good at defining contextualized words and new words; reverse dictionary models solve the "tip-of-the-tongue" problem and they are useful for paraphrasing and semantic search.[7] The technology can also be integrated into downstream applications for language education, writing assistance, etc.

Building on the insights we learned from Chapter 3, we put forward a perspective that a contextualized word-definition pair resembles a parallel sentence pair. The texts on both ends share the same meaning in both cases, but a word and its definition differ in surface texts and the parallel sentences in language. The parallelism is not affected if the input and output ends are swapped. While previous works solve one problem at a time, we argue that both reverse dictionary and definition generation can be learned concurrently, using a single embedding space bottleneck to train enhanced semantic representations inspired by the work of LASER (Artetxe and Schwenk, 2019a).

Based on this motivation, we develop a neural network that can embed words and definitions into the same semantic space and generate both forms. By treating words and definitions as different "languages", we force all semantic information to be retained in the bottleneck module. This chapter further explores the opportunity of self-data augmentation by training on the same copy of data but in multiple ways simultaneously, hoping to learn better meaning representations. Specifically, our training paradigms include the two original tasks, the reconstruction (autoencoding) of words and definitions, as well as embedding similarity. Such a system can be viewed as a neural dictionary that supports two-way indexing and querying.

This chapter discusses how we utilize self-augmentation—by combining training data in various ways—to enable multiple training objectives in a unified model for reverse dictionary and definition modelling. Experiments investigate different data availability, languages, and embedding architectures, and our data modelling brings in

[7] https://www.onelook.com/

substantial performance improvement. We use an adversarial system to report the difficulty in evaluating definitions as well as investigate various input and output features. Our system implementation is publicly available.[8]

4.2 Related Work

Although research on the two tasks can be traced back to the early 2000s, recent research has shifted towards neural networks for performance considerations. We describe the neural methods that tackle reverse dictionary and definition modelling separately below. From a model architecture perspective, a key difference is that previous works often use either an encoder (for reverse dictionary) or a decoder (for definition generation), but ours is an encoder-decoder model to accommodate the unified training with augmented data.

4.2.1 Reverse dictionary

Hill et al. (2016b) pioneered the use of recurrent neural networks (RNN) and bag-of-words models to convert texts to word vectors, on top of which Morinaga and Yamaguchi (2018) added an extra word category classifier. Pilehvar (2019) integrated super-sense into target embeddings to disambiguate polysemous words. Zheng et al. (2020) designed a multi-channel network to predict a word together with its linguistic features like category, POS tag, morpheme, sememe, etc. We find that most previous works attempt to include as much linguistic information as possible to improve the performance of the reverse dictionary task. Although these works have achieved promising results, they rely on and thus are limited by the availability of linguistic annotations.

As a notable difference from previous papers, and as a novel approach to the modelling of words and definitions, our work tackles the problem via self-data augmentation for improved representations, without the need for extra annotated resources. The proposed framework learns shared hidden representations for definitions and words with the same meaning. From this aspect, Bosc and Vincent (2018)'s work is related. They trained word embeddings via definition reconstruction. Our work has also been largely inspired by Schwenk (2018)'s idea of obtaining multilingual sentence embeddings as introduced in Chapter 3.2: they train a multilingual encoder by embedding a

[8] https://github.com/PinzhenChen/unifiedRevdicDefmod

sentence in any language in a shared space, which is then used as an input to generate a parallel sentence in another language. Our work shares a common philosophy that a common space can be used to represent words and definitions. We make a change by replacing LSTM modules with Transformers in encoders and decoders which are popular backbones currently, as well as incorporating an embedding similarity loss that aligns with the criterion used in the downstream retrieval task.

4.2.2 Definition modelling

Noraset et al. (2017) first used RNNs for definition generation, and they explored different model architectures and input features. This is followed by Gadetsky et al. (2018) who added attention and word context, as well as Chang et al. (2018) whose model projects words and contexts to a sparse space, then generates from selected dimensions only. Mickus et al. (2019)'s proposed model encodes a context sentence and uses a mark on the word of interest to be defined. Recently, Bevilacqua et al. (2020)'s work enabled a model to define a flexible span of words as opposed to a single word. Apart from generating definitions freely, Chang and Chen (2019) took a new perspective of re-formulating the generation task to definition retrieval from a dictionary. This approach eases the difficulty of the generation process, but it is less extensible since it requires a dictionary to be accessible in advance.

4.3 Methodology

4.3.1 A unified model

A word and its definition share the same meaning, even though they exist in different surface forms. When modelling their semantics using a neural method, we hypothesize that a word and its definition can be encoded into the same representation space, similar to how LASER compresses sentences in different languages using a single encoder bottleneck (Schwenk, 2018). In this chapter, we present an adapted and improved architecture for word-definition joint modelling: a model that maps both words and definitions into a shared space; the hidden states in the shared space can be converted into words and definitions freely. Essentially, the shared representation can be viewed as an autoencoding of the meaning shared by a word and its definition.

We illustrate our proposed architecture in Figure 4.1. There are three types of

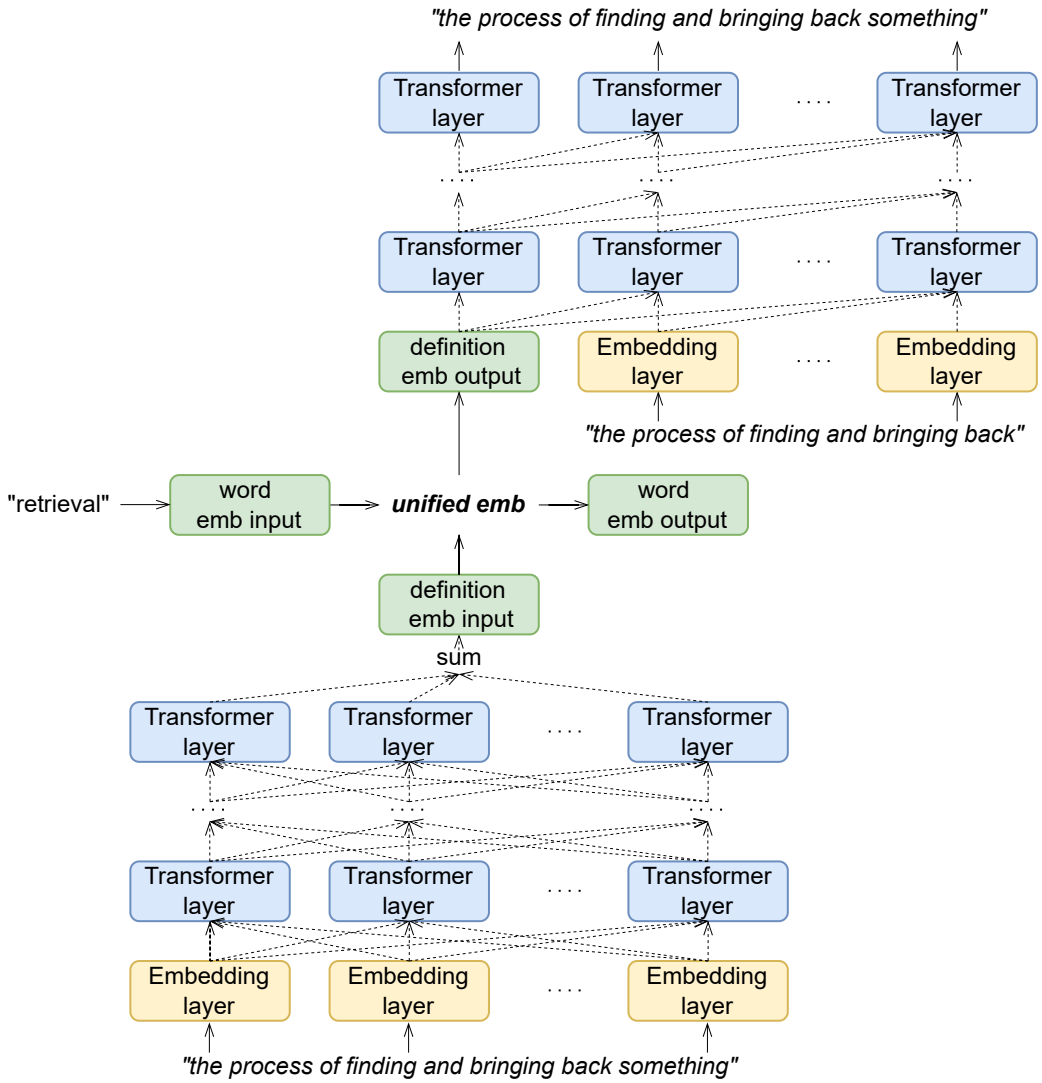


Figure 4.1: An illustration of our unified model.

modules: embedding layers (yellow), Transformer modules self-attention (blue), as well as linear layers for up- and down-projections (green). The transformer modules and linear layers are inter-connected by a unified embedding space in four ways:

1. The "word emb input" module is a linear layer $L_{\text{word\_in}}$ that can project any input word or word embedding into the unified space.

2. Linear layer $L_{\text{word\_out}}$, represented by "word emb output", can take an embedding in the unified space as input and produce an output word embedding.

3. The Transformer encoder T_{enc} is made up of the embedding layer and stacked transformer layers with self-attention. It embeds definition tokens, and then the

token embeddings are added up. The summed state passes through "definition emb input", which is a linear layer $L_{\text{def\_in}}$ that produces a final embedding in the unified space.

4. An embedding in the unified space is transformed through "definition emb output" layer $L_{\text{def\_out}}$ and then fed into the Transformer decoder T_{dec} as the begin-of-sentence token to generate a definition. The decoder has masked self-attention to prevent information leaks from future time steps.

A difference between our model and a conventional Transformer encoder-decoder model is that we do not allow for encoder-decoder attention. Without it, the definition decoder can only retrieve information from the shared space (denoted as "unified emb" in Figure 4.1); word and definition information is "bottlenecked" through the shared space before they can be used for generation. Having the four-way connection, the model is capable of encoding words and definitions as well as generating word embeddings and definitions.

Under our framework, compared to training either reverse dictionary or definition generation, jointly modelling both will require more GPU memory due to a larger model size. Nonetheless, it does not increase the model size at inference time for each task, because for a single task only one of the encoder or the decoder needs to be used.

4.3.2 Self-augmentation and multi-tasking

We apply a self-augmentation method to the genuine training data without any external resources, that is compatible with the unified model training. From a word-definition pair (X,Y), we additionally create word-word (X,X), definition-word (Y,X), and definition-definition (Y,Y) pairs by re-arranging the data directionality. The resulting data is four times larger than the original data and of high quality since it comes from a genuine source.

Note that words first need to be converted into embeddings. The augmented data is then used to train the unified model in four directions:

1. Definition modelling using (X,Y): the word X is input via $L_{\text{word\_in}}$ and the output definition is obtained from the decoder T_{dec}. The definition modelling objective computes a token-level loss $\mathcal{L}_{\text{defmod}}$ between $T_{\text{dec}}(L_{\text{def\_out}}(L_{\text{word\_in}}(X)))$ and the definition Y.

2. Reverse dictionary using (Y,X): the definition Y is encoded by T_{enc} and then mapped to the unified space by followed by $L_{\text{def\_in}}$, and the output word is obtained from $L_{\text{word\_out}}$. An embedding-level similarity loss $\mathcal{L}_{\text{revdic}}$ between $L_{\text{word\_out}}(L_{\text{def\_in}}(T_{\text{enc}}(Y)))$ and the word X is computed.

3. Definition autoencoding using (Y,Y): the definition Y is encoded by T_{enc} and then mapped to the unified space by $L_{\text{def\_in}}$. The output definition is obtained via $L_{\text{def\_out}}$ followed by the decoder T_{dec}. The definition autoencoding an token-level loss $\mathcal{L}_{\text{defAE}}$ between $T_{\text{dec}}(L_{\text{def\_out}}(L_{\text{def\_in}}(T_{\text{enc}}(Y))))$ and the definition Y itself. Further to vanilla autoencoding, we use two token corruption schemes from denoising autoencoding: each definition token has a small probability of being replaced by a random token or a mask token.

4. Word autoencoding using (X,X): the word is mapped into the unified space by $L_{\text{word\_in}}$ and then mapped back to its original space by $L_{\text{word\_out}}$. An embedding similarity loss $\mathcal{L}_{\text{wordAE}}$ between $L_{\text{word\_out}}(L_{\text{word\_in}}(X))$ and the word X itself is used to update the model.

In addition to the four objectives above, we also consider an embedding similarity loss $\mathcal{L}_{\text{embsim}}$ between the shared space representation of a word $L_{\text{word\_in}}(X)$ and the representation of its definition $L_{\text{def\_in}}(T_{\text{enc}}(Y))$. This explicitly encourages the alignment of a word and its definition in the unified space. Finally, our training procedure minimizes the overall loss $\mathcal{L}_{\text{unified}}$ that adds up all above losses with equal weights:

$$\mathcal{L}_{\text{unified}} = \mathcal{L}_{\text{defmod}} + \mathcal{L}_{\text{revdic}} + \mathcal{L}_{\text{defAE}} + \mathcal{L}_{\text{wordAE}} + \mathcal{L}_{\text{embsim}}$$

In our experiments, the token-level losses ($\mathcal{L}_{\text{defmod}}$ and $\mathcal{L}_{\text{defAE}}$) are selected to be *cross-entropy*; we explore two options for the embedding losses ($\mathcal{L}_{\text{revdic}}$, $\mathcal{L}_{\text{wordAE}}$, and $\mathcal{L}_{\text{embsim}}$): *cosine similarity* as well as *mean squared error*.

4.3.3 Word sense disambiguation

A word is often associated with multiple definitions due to the presence of polysemy, sense granularity, etc. In our practice, the one-to-many word-definition relationship does not harm reverse dictionary, since our model can master mapping different definitions into the same word vector. However, this is problematic for definition modelling, as telling the exact word sense without context is difficult if a word is polysemous. Our approach alleviates this problem by embedding a word together with its context

using BERT and generating a definition from the corresponding output word embedding from BERT (Devlin et al., 2019). We sum up the resulting sub-word embeddings for each word if a word to be defined is segmented by the BERT tokenizer.

4.4 Experiments on English

We first benchmark our unified model on reverse dictionary and definition modelling tasks using existing English datasets which are relatively large. We compare our model using multi-way data augmentation with baseline Transformers as well as models from previous papers.

4.4.1 Data and evaluation

HILL We evaluate our unified model on the English reverse dictionary data Hill et al. (2016b) created. There are roughly 100K words and 900K word-definition pairs. There are three official test splits to test a dictionary system's memorizing and generalizing capabilities:

1. *seen* test: 500 word-definition pairs drawn from the training data.

2. *unseen* test: 500 word-definition pairs that do not have any word or definition present in the training set.

3. *human description* test: 200 words that are described and quality-assured by human annotators as opposed to being existing dictionary definitions. These test definitions should not have been seen by any system, including online systems like OneLook.com which might have ingested the unseen test split from a dictionary.

A reverse dictionary system is supposed to read a definition and then generate an output word representation for it. All candidate word representations are then ranked based on similarity to the model output. The evaluation metrics are retrieval accuracies at rank cutoff at 1, 10 and 100 (acc@1/10/100), as well as the median (med.) and standard deviation (std.) of the correct target words' ranks. Previous papers might use "standard deviation" or "rank variance", but our discussions stick to standard deviation for consistency.

CHANG We test our model's definition modelling performance using Chang and Chen (2019)'s data from the Oxford English Dictionary. Each data entry is a tuple of a word, its usage (context), and its definition. The data has two splits: *seen* and *unseen*. The training split we use consists of 530K training instances, and the *unseen* test set consists of 1K words paired with 16K definitions and their context. Generated definitions are scored against the references using corpus-level BLEU-4 implemented in NLTK, and ROUGE-L F1[9] (Papineni et al., 2002; Lin, 2004; Bird et al., 2009).

The questionable seen test split Understandably, a dictionary needs to "memorize" word entries, so both HILL and CHANG supply a *seen* test split drawn from the training data. However, we argue that this is uninformative for benchmarking deep learning-based systems, for it implicitly encourages overfitting. We have observed this in some past work. On the other hand, the foremost function of a neural dictionary should be to deal with unseen, contextualized words and definitions; otherwise, a traditional rule-based system suffices. We, hence, omit evaluation and comparisons on the *seen* test split, and we advocate that future research should not focus on it either.

4.4.2 System configurations

Transformer baselines Prior to our work, many studies have used LSTM modules as the backbone of their dictionary systems (e.g. Zheng et al., 2020). Given the widespread use of Transformer models in our field, we are curious to test it as our baselines. For the reverse dictionary task, a Transformer encoder is used to encode the input definition into a representation; for definition generation, a Transformer decoder is used.

Our unified model Our proposed model connects and trains the above two baselines with extra projection layers as shown in Figure 4.1 in green. The shared representation has the same size as the input embeddings and a residual connection around it (He et al., 2016). As an additional variant, we tie the Transformer encoder and decoder's embedding and output layers (Press and Wolf, 2017) (denoted as the "embedding layer" in the encoder and decoder as well as the output layer of the decoder in Figure 4.1). This is only possible with our proposed multi-task framework since a Transformer encoder or decoder alone does not have both encoder and decoder embeddings.

[9] https://github.com/pltrdy/rouge

It is worth noting that we train separate unified models on HILL and CHANG data to evaluate reverse dictionary and definition modelling respectively. This is for the purpose of maintaining fair comparisons with previous works, which would only use one of the two datasets for each task.

Hyperparameters We adjusted the hyperparameters for the baseline using the validation set and kept the values unchanged for the proposed model which joins the baseline Transformer modules. We list all hyperparameters in Table 4.1 and highlight the selected ones in boldface if multiple values were tested. The trial is carried out one by one for each hyperparameter. We use whitespace to tokenize definitions in the training data into an open vocabulary. We choose cross-entropy as the token loss and mean squared error (MSE) as the embedding loss. On a single Nvidia GeForce GTX 1080 Ti, it takes 60 hours for our model to converge on the HILL dataset. A model trained on the CHANG dataset converges after 6 hours on a single Nvidia GeForce RTX 2080 Ti.

Model comparisons For reverse dictionary, our comparison includes the following models: OneLook.com, bag of words (Hill et al., 2016b), RNN (Hill et al., 2016b), word category inference (Morinaga and Yamaguchi, 2018), multi-sense (Kartsaklis et al., 2018), super-sense (Pilehvar, 2019), BiLSTM (Zheng et al., 2020) and multi-channel (Zheng et al., 2020). Systems' results are reported as in Zheng et al. (2020)'s re-run, so following their practice to ensure a fair comparison, we embed the target words with 300d `word2vec` (Mikolov et al., 2013), but input definition tokens are randomly initialized as 256d weights and trained from scratch.

In definition modelling comparison. We include RNN (Noraset et al., 2017) and xSense (Chang et al., 2018) as replicated by Chang and Chen (2019), but we omit Chang and Chen (2019)'s results. This is because their model retrieves definitions from pre-written dictionary entries. To some extent, this is an easier "oracle" experiment—instead of freely generating a definition, the model selects one from a pool of definitions. Input words to be defined are embedded by the 768d `BERT-base-uncased` (Devlin et al., 2019) with a usage example to provide context-specific information, while target-side definition token embeddings are randomly initialized.

Hyperparameter	Value
learning rate	1e-3, **1e-4**, 1e-5 and 1e-6
optimizer	Adam (Kingma and Ba, 2015)
beta1, beta2	0.9, 0.999
weight decay	1e-6
token loss	cross-entropy
embedding loss	**MSE**, cosine (failed to converge)
training batch size	HILL: 256
	CHANG: 128
stopping criterion	5 non-improving validations
decoding beam size	**6**, 64
Transformer depth	**4**, 6
Transformer head	**4**, 8
Transformer dropout	0.1, **0.3**
definition token corruption	**0**, 0.1
linear layer dropout	0.2
linear layer dimension	HILL: 256
	CHANG: 768
shared layer dimension	HILL: 256
	CHANG: 768
trainable parameters	HILL: 35.1M
	CHANG: 62.7M

Table 4.1: Model and training configurations.

4.4.3 Automatic metric results

Reverse dictionary We report the results of our baseline and unified models in Table 4.2. We also include a variant of the unified model where embedding layers and the decoder output layer share the same set of weights, as denoted by "+ share embed". We observe that the baseline Transformer encoder already achieves solid numbers which surpass many previous works in terms of the median rank and accuracy at 100.

Our proposed unified training significantly improves upon the baseline: the median rank is four to five times smaller, and accuracies at 1 and 10 are three to four times

System	unseen test				human description test			
	med. rank	acc@ 1/10/100	rank std.†	real std.	med. rank	acc@ 1/10/100	rank std.†	real std.
OneLook.com	-	-	-	-	5.5	.33/.54/.76	332	-
bag of words	248	.03/.13/.39	424	-	22	.13/.41/.69	308	-
RNN	171	.03/.15/.42	404	-	17	.14/.40/.73	274	-
category inference	170	.05/.19/.43	420	-	16	.14/.41/.74	306	-
multi-sense	276	.03/.14/.37	426	-	1000	.01/.04/.18	404	-
super-sense	465	.02/.11/.31	454	-	115	.03/.15/.47	396	-
BiLSTM	101	.07/.24/.49	401	-	5	.25/.60/.83	214	-
multi-channel	54	.09/.29/.58	**358**	-	**2**	**.32**/.64/.88	203	-
Transformer encoder	79	.01/.14/.59	473	125	27	.05/.23/.87	332	49
unified	**18**	**.13/.39/.81**	386	**93**	4	.22/.64/**.97**	**183**	**30**
+ share embed	20	.08/.36/.77	410	99	4	.23/**.65/.97**	**183**	32

†Zheng et al. set a rank to 1000 if larger than 100; we follow suit and also list the actual std.

Table 4.2: Reverse dictionary results on the test splits from HILL.

higher. Compared to the previous state-of-the-art "multi-channel", we obtain better median rank and accuracies on *unseen* definition-word pairs. On *human descriptions* our models yield comparable median rank and accuracies; in terms of standard deviation, the unified model is remarkably better than the Transformer baseline and many previous systems, indicating a consistent performance across different words and definitions.

In general, previous works usually make improvements by modelling added linguistic information, for instance, "category inference", "multi-sense", "super-sense", and "multi-channel". The highlight of our work is that it attains a superior position with self-contained data augmentation, without the need for additional signals. The only exception is a word embedder, but this is required for word representation comparison and is also used in previous research. Consequently, our approach could be a more generic framework for this task that is not limited by the availability of linguistic resources.

System	unseen test	
	BLEU	ROUGE-L
RNN	1.7	15.8
xSense	2.0	15.9
Transformer decoder	2.4	17.9
unified	2.2	18.5
+ share embed	**3.0**	**20.2**

Table 4.3: Definition modelling results on the CHANG data.

Definition modelling The numbers for definition generation are listed in Table 4.3. Similar to previous trends, our Transformer decoder establishes a stronger mark in comparison with previous work under both BLEU and ROUGE. Our unified model achieves similar BLEU and slightly better ROUGE. With tied embeddings, we observe a substantial improvement and the model achieves state-of-the-art scores. Nonetheless, we point out that, while ROUGE-L scores look reasonable, the single-digit BLEU might hint at the poor quality of the generation. It would be difficult to qualify the scores and differentiate the two systems if the BLEU numbers are too low. Hence, we discuss the human evaluation of definitions in the next section.

4.4.4 Human judgement on definitions

Supplementary to the automatic evaluation for definition generation, we conduct both reference-free and reference-based human evaluations to distinguish two systems: the Transformer decoder baseline and our unified model with shared embeddings. The two evaluation settings differ in the "anchors" shown to the annotators:

- *Reference-free*: a human annotator is asked to pick the output definition they prefer after seeing the input word without the reference definition.
- *Reference-based*: a human annotator is asked to pick the output definition they think is closer to the reference dictionary definition without seeing the input word.

We sampled test instances and presented the two systems' corresponding outputs in a random order to the human evaluators. Two annotators in total evaluated 80 test

instances for each evaluation setting. They were required to make a decision between the two outputs unless the outputs were the same. Table 4.4 records the number of times each model's output is favoured over the other. The columns do not add up to 80 (or 100%) because we do not count the cases where both systems generated the same output.

System	Human Preference	
	Reference-free	Reference-based
Transformer decoder	25 (31%)	32 (40%)
unified + share embed	**50 (63%)**	**42 (53%)**

Table 4.4: Chances a model's output is preferred by human evaluators.

According to Table 4.4, regardless of the evaluation condition, evaluators often regard the unified model's outputs as better. Especially in the reference-free scenario, which resembles a real-life application of definition generation, our unified model wins notably with a doubled preference count. Through both previous automatic metric scores and human evaluation, we conclude that our unified model is a better definition generator than a vanilla Transformer decoder.

4.4.5 Vocabulary and shared embeddings

Vocabulary We have used an open-ended vocabulary by including all tokens separated by whitespace in the training data. It might be argued that it comes with weaknesses like being oversized and not being able to represent unknown tokens at test time. We are interested in understanding if the generation process can be enhanced with a closed vocabulary. To this end, we train a model with a 25K unigram SentencePiece vocabulary (Kudo and Richardson, 2018). All other configurations remain the same as the best-performing unified model with shared embeddings. We observe that BLEU and ROUGE-L drop to 2.5 and 18.7 respectively from our best performance in Section 4.4.3. This implies that using an open vocabulary is not an issue in our earlier experiments. We argue that a closed vocabulary is not a crucial requirement in our case potentially because:

- During definition modelling, this vocabulary is solely used for the generation, while word encoding does not require it, so the input end does not suffer from the unknown word problem.

- It is likely sufficient to produce a definition by combining existing words from definition words in the training data since definition generation is not an overly creative task,
- On the other hand, during reverse dictionary and definition autoencoding, unseen words in an input definition will be represented by an unknown token. This is effectively equivalent to word-level masking, which acts as a denoising autoencoding practice.

Shared embeddings For definition modelling, sharing the embedding and output layer weights brings significant improvement to our proposed approach, but in the reverse dictionary task, our model is observed to arrive at desirable results without it. We conjecture that this is because well-trained embedding and output layers particularly benefit language generation (Press and Wolf, 2017) but might be less beneficial to the encoder side. It further indicates the usefulness of our unified approach whereby all embedding and output layers can be weight-tied, enabled by simultaneously training the two Transformer modules with the augmented data with multiple tasks.

4.5 Extensions to More Languages and Embeddings

4.5.1 Overview

Using our proposed model, we participated in an international shared task held at SemEval 2022 on both reverse dictionary and definition generation (Mickus et al., 2022). The task features data in five languages and three embedding types but with a relatively modest size. This section describes the extensibility of our data and modelling approaches in these scenarios. Our model performance was evaluated by the task organizers and overall we won the most number of tracks in the shared task. We also express our opinions on the difficulty of evaluating generated definitions.

4.5.2 Datasets

The organizers provide datasets for five languages: English (`en`), Spanish (`es`), French (`fr`), Italian (`it`), and Russian (`ru`). For each language, the data is split into train, validation, test, and trial sets, at sizes 43.6k, 6,4k, 6.2k, and 0.2k. The test set was only made accessible to the public after the official evaluation of submissions had been

done. Also, 256d word embeddings trained using three different architectures were supplied:

- `sgns`: statically contextualized embeddings learned with skip-gram with negative sampling, one variant of `word2vec` (Mikolov et al., 2013). This embedding technique does not account for polysemous words—a word in different contexts will have the same embedding vector.

- `char`: embeddings from an autoencoder trained on the character-level spelling of a word. Similarly, the embedding is the same regardless of a word's precise sense in a context.

- `electra`: Transformer-based embeddings from a generator-discriminator model following Clark et al. (2020). A word is embedded within a specific context, so the same word appearing in different contexts will receive different embedding values.

Embeddings from `electra` are not available for `es` and `it`; despite this, the data still covers 13 language-embedding combinations. All embedding architectures are trained on comparable corpora for all languages.

For a fair comparison, it was required that only word embeddings and definition glosses could be used for model training. Words are provided as embeddings rather than text strings. The actual words are not accessible by participants so no extra resources can be used. This poses an underlying challenge for the systems we compared to in previous sections because many of these are linguistic resource-dependent. On the other hand, our data augmentation is self-contained, so we can assess our approach in a low-resource setting involving multiple languages and embedding types.

Human annotations are included in the trial split for experimental analysis. The snippet below exemplifies a single data instance with all possible fields. The train, validation, and test sets consist of only the key-value pairs in boldface; all fields are found in the tiny trial set. We include an example of an entry in the trial set with fields commented.

{"**id**": "en.trial.2",
"**sgns**": [2.08729, 0.26177, ...],
"**char**": [0.38789, 0.19716, ...],
"**electra**": [-1.47715, -0.47424, ...],
"**gloss**": "A mixture of other substances or things.",

```
“word”: “cocktail”,
“pos”: “noun”,                                  # POS tag/word category
“example”: “a cocktail of illegal drugs”,       # word usage within context
“type”: “hypernym-based”,
“counts”: 4187,                                 # number of appearances in a corpus
“f_rnk”: 13245,                                 # frequency rank
“concrete”: 1,
“polysemous”: 0}
```

4.5.3 Evaluation metrics

There are two tracks: reverse dictionary and definition modelling. For the reverse dictionary task, there are separate sub-tracks for each language-embedding combination. For definition modelling, there is a sub-track for each language where all embeddings can be utilized simultaneously.

Multiple evaluation metrics are used. Reverse dictionary system outputs are evaluated with three:

1. *MSE*: mean squared error between references and generated embeddings.
2. *cosine*: cosine similarity between references and generated embeddings.
3. *ranking score*: a percentage score measuring how many other test instances have a higher cosine similarity with a generated embedding than its reference does.

The definition modelling performance is measured by three metrics too:

1. *sense-BLEU*: sentence-BLEU implemented in NLTK with smoothing method 4 (Papineni et al., 2002; Chen and Cherry, 2014).
2. *lemma-BLEU*: the maximum sense-BLEU between a generated gloss and all definitions of the input word corresponding to the same part of speech.
3. *MoverScore*: a neural distance measure based on multilingual BERT (Zhao et al., 2019).

All evaluation scripts are provided by the task organizers and evaluation is carried out by the shared task organizers.

4.5.4 Experimental setup

Our experiment configurations follow the previous experiments in Section 4.4. Definition glosses are tokenized by whitespaces into an open vocabulary. The word embedding layers in the Transformer modules are initialized randomly, due to the reason

that we cannot use pre-trained embedders for definitions for fair comparison. For input words, we use the embeddings as provided. Loss functions are cross-entropy for tokens and MSE for embeddings.

Regarding definition modelling, we do not combine various embeddings as input since our model unifies the reverse dictionary and definition generation training for a particular embedding type; this might put us at a disadvantage in the team ranking. We apply the same configurations to all language-embedding combinations. Training a unified model on an Nvidia GeForce RTX 2080 Ti takes roughly three hours. Training stops after five non-improving validations. The beam size is set to 6 for definition decoding. Hyperparameters are the exact same as Table 4.1, except for linear layer shape which is fixed to the word embedding size—256 in this case.

4.5.5 Ensembling for reverse dictionary

Ensembling is a common technique to enhance machine learning performance. Specifically for reverse dictionary, to build a competitive system, we perform average ensembling: for each test instance, its final word embedding is obtained by averaging all the corresponding output embeddings from different models. We ensemble up to 21 models, of the same unified architecture, trained with various random seeds.

4.5.6 A handcrafted n-gram baseline for definition modelling

Upon our initial trial set inspection of definition modelling, the generations are mostly meaningless, scoring a low sense-BLEU of about 3. To understand how indicative BLEU is in low-resource definition modelling, we handcraft a nonsensical *n*-gram submission. The rule is that for each test instance, we simply concatenate the most frequent bigram with the most frequent unigram, computed on all definitions in the training data. The phrases we prepare for each language are:

en: , or .
es: de la .
fr:) (.
it:) (.
ru: в . ,

4.5.7 Results

We did not have access to the test references before the official evaluation was completed, so the results in this section are taken directly from the official leaderboard.

Reverse dictionary Our reverse dictionary test scores are reported in Table 4.5, Table 4.7, and Table 4.8 for word retrieval based on `electra`, `sgns` and `char` respectively. The suggested baseline from the shared task was a Transformer encoder model; we reproduced it and compared it with our unified model. Overall, our unified model achieves the same (2/39 cases) or better (37/39 cases) numbers compared to the baseline, across all metrics, all languages, and all embedding types. This clearly proves the strength of our multi-way data modelling approach. Additionally, we made ensemble submissions from 17 and 21 models for the reverse dictionary task. It is found that ensembling usually further improves word retrieval when the embedding method is `char` or `electra`, but it does not seem to significantly help `sgns`.

System	en			fr			ru		
	MSE	cosine	rank	MSE	cosine	rank	MSE	cosine	rank
Transformer	1.34	0.842	0.497	1.18	0.853	0.497	0.898	0.718	0.498
unified	1.32	0.844	0.495	1.08	0.861	0.476	0.846	0.731	0.421
ensemble-17	**1.31**	**0.847**	**0.490**	**1.07**	**0.862**	**0.479**	**0.829**	**0.735**	**0.417**
ensemble-21	**1.31**	**0.847**	0.491	**1.07**	0.861	0.480	**0.829**	0.734	0.419

Table 4.5: Reverse dictionary test performance with **`electra`** as target embeddings, in MSE (↓), cosine similarity (↑), and ranking score (↓).

Definition modelling As per Table 4.9, out of 39 comparisons with the Transformer decoder baseline recommended in the shared task, the unified model wins 28 times and loses 11 times, which marks our method as favourable. We also include our handcrafted n-gram baseline to probe for reasonable random performance. Interestingly, it surpasses genuine models on English BLEU scores and even tops in French sense-BLEU among all shared task participants. This implies that either BLEU scores are not informative or the submissions are as embarrassing as the n-grams. On the contrary, MoverScore is always effective in downing the n-grams. Sadly, our manual inspection

reveals that most model-generated glosses are inaccurate. The dissatisfying results might be due to the modest training size.

Model	Embedding	Output Definition
gold	n/a	A mixture of other substances or things.
unified	sgns	In a manner.
	char	A person or thing.
	electra	The act of something.

Table 4.6: The gold and model-generated definitions for "*cocktail*" in the context of "*a cocktail of illegal drugs*".

Sanity check on sense-BLEU Given the observed problem with the evaluation of generated definitions, we design an extra sanity check on the sense-BLEU metric. For the English trial set embedded with `sgns`, we remove punctuation marks and NLTK-defined stop words from references, as well as from our unified model's generations. Sense-BLEU drops from 3.31 to 0.39, and it further worsens to 0 if we disable sentence-level smoothing. Evidently, sense-BLEU and thereby lemma-BLEU are hugely inflated by functional tokens as well as smoothing. This implies that the evaluation of definition generation may not accurately differentiate model quality, so we should not rely on BLEU scores to compare models.

We manually skimmed through the generated definitions from our models and unfortunately, we found that the generations are not yet usable. In Table 4.6, we list an example of such. Nonetheless, results from the reverse dictionary track should still be indicative as the embedding distance scores genuinely reflect how accurate or close a retrieved word is, with respect to a gold word given an input definition.

System	en			es			fr			it			ru		
	MSE	cosine	rank	MSE	cosine	rank	MSE	cosine	rank	MSE	cosine	rank	MSE	cosine	rank
Transformer	0.884	0.189	0.439	0.905	0.241	0.462	1.06	0.275	0.360	1.10	0.245	0.451	0.561	0.295	0.432
unified	0.871	**0.241**	**0.326**	0.868	0.339	**0.271**	**1.03**	**0.312**	**0.302**	1.05	0.371	**0.197**	0.553	0.327	0.340
ensemble-17	**0.864**	0.225	0.374	**0.860**	**0.347**	**0.271**	**1.03**	0.305	0.334	**1.03**	0.373	0.206	**0.538**	0.381	0.251
ensemble-21	0.865	0.225	0.374	**0.860**	**0.347**	**0.271**	**1.03**	0.306	0.330	**1.03**	**0.374**	0.205	**0.538**	**0.383**	**0.247**

Table 4.7: Reverse dictionary test performance with **sgns** as target embeddings, in MSE (↓), cosine similarity (↑), and ranking score (↓).

System	en			es			fr			it			ru		
	MSE	cosine	rank	MSE	cosine	rank	MSE	cosine	rank	MSE	cosine	rank	MSE	cosine	rank
Transformer	0.161	0.795	0.500	0.551	0.820	0.499	0.404	0.764	0.495	0.400	0.720	0.499	0.144	0.829	0.496
unified	0.143	0.795	0.500	0.480	0.834	0.431	0.347	0.782	0.448	0.337	0.745	**0.428**	0.119	0.849	0.395
ensemble-17	**0.142**	0.795	0.500	**0.467**	**0.839**	**0.424**	0.336	0.788	0.429	**0.334**	**0.747**	0.429	**0.116**	0.851	0.390
ensemble-21	**0.142**	0.795	0.500	**0.467**	**0.839**	0.425	**0.335**	**0.789**	**0.428**	**0.334**	**0.747**	0.429	**0.116**	**0.852**	**0.389**

Table 4.8: Reverse dictionary test performance with **char** as target embeddings, in MSE (↓), cosine similarity (↑), and ranking score (↓).

Source embed.	System	en			es			fr			it			ru		
		Mover Score	sense- BLEU	lemma- BLEU	Mover Score	sense- BLEU	lemma- BLEU	Mover Score	sense- BLEU	lemma- BLEU	Mover Score	sense- BLEU	lemma- BLEU	Mover Score	sense- BLEU	lemma- BLEU
n/a	*n*-gram baseline	-0.004	**3.06**	**3.81**	-0.032	2.73	3.67	-0.176	**2.95**	3.56	-0.164	1.89	2.74	-0.006	2.65	3.31
sgns	Transformer	0.100	2.91	3.67	0.088	**3.47**	**5.28**	-0.019	2.34	3.38	0.046	4.62	6.97	**0.109**	**4.91**	7.14
	unified	0.098	3.01	3.80	**0.101**	3.42	5.14	-0.064	1.59	2.38	**0.107**	**6.01**	**9.17**	0.095	4.59	6.82
char	Transformer	0.101	2.47	3.02	0.064	2.06	2.88	-0.186	0.11	0.11	0.019	2.09	2.99	0.092	4.01	5.87
	unified	**0.104**	2.83	3.40	0.065	2.14	2.96	**0.026**	2.42	**3.82**	0.044	2.93	4.29	0.085	4.80	**7.24**
electra	Transformer	0.070	2.53	3.26	n/a			-0.075	1.38	1.93	n/a			0.090	3.78	5.45
	unified	0.094	2.75	3.43				-0.045	1.60	2.29				0.088	4.08	5.86

Table 4.9: Definition modelling test performance measured by MoverScore (↑), sense-BLEU (↑), and lemma-BLEU (↑).

4.6 Analysis on Embeddings, Languages, and Features

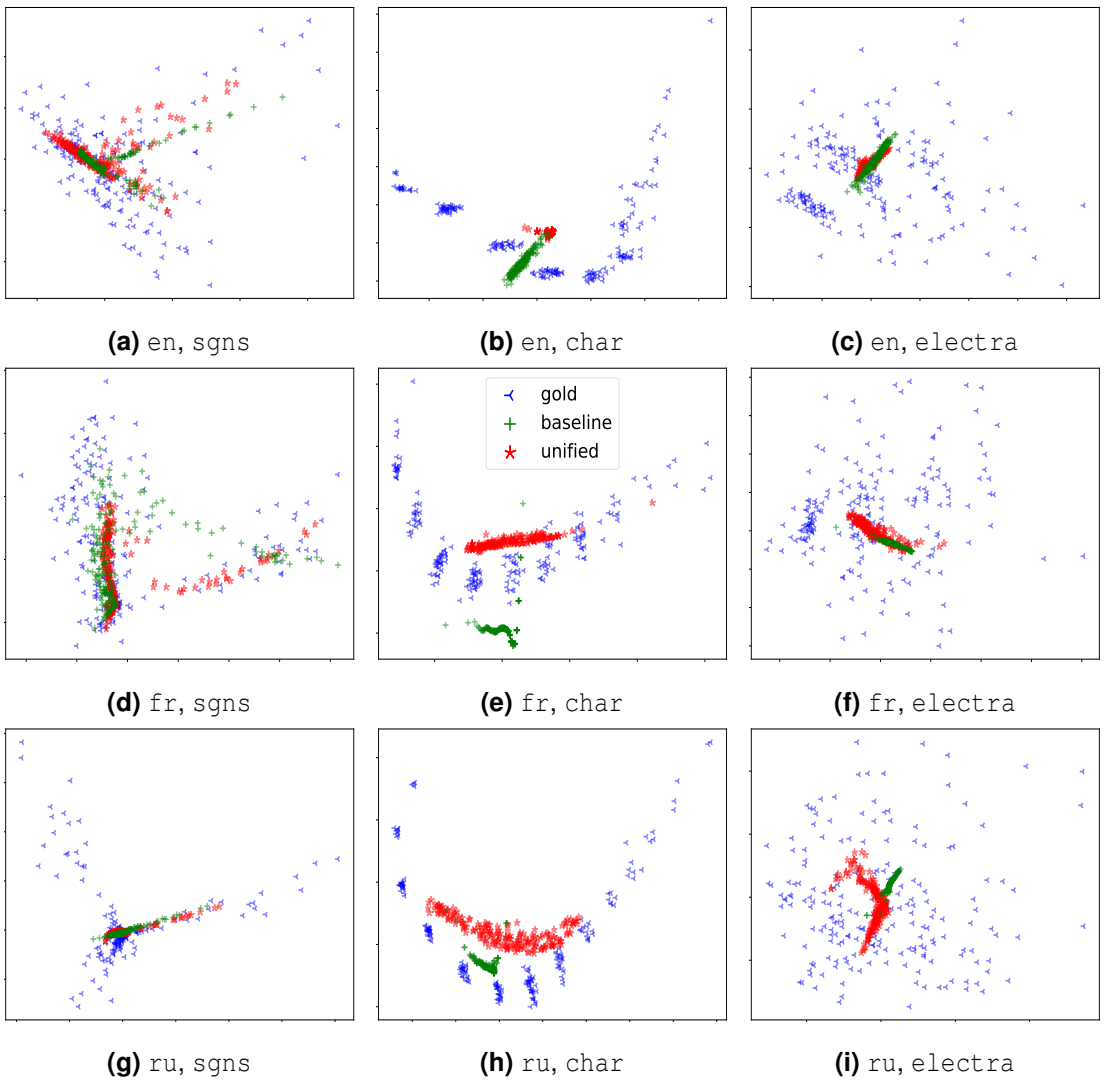
Following the previous experiment section, we try to understand the influence of embedding architectures, languages, and linguistic features on word retrieval and definition generation performance for our joint modelling approach.

4.6.1 Embedding types

Reverse dictionary Absolute metrics like MSE and cosine similarity are not comparable across different embedding types since a retrieval method based on these considers the relative margin. On the other hand, ranking scores can indicate how close a generated embedding is to the gold than to other false candidates. A lower ranking score is preferred for the accurate indexing and retrieving of a word. Random baseline scores should be around 0.5 which means that the gold is ranked in the middle of all candidates. Across Tables 4.8 and 4.5, we find that most `char` and `electra` ranking scores fall between 0.4 and 0.5, which is not extremely ideal. On the other hand, as Table 4.7 suggests, `sgns` is more useful; the baseline's ranking scores start at around 0.45, and our models can improve these up to 0.25.

We run principal component analysis (PCA) to reduce the gold and output embeddings to 2 dimensions to retain only the most important components in the embedding. It also makes visualization easier. In Figure 4.2, we present three languages: English, French, and Russian, which come with all embeddings. The unified model usually outputs to a larger space than the baseline, hinting at a positive correlation between output spread and performance. Gold `electra` has the most isotropic space, but neither the baseline nor the unified model could imitate the distribution. `Char` has a crescent shape with several clusters inside, which is unlikely to be cosine-friendly—potentially the space needs to be transformed before cosine can be used as a retrieval metric. These problems are alleviated on `sgns`, which witnesses the best ranking scores.

Definition modelling The `sgns` architecture is again the winner, as models trained with `sgns` reach the top in many metrics. `Char` is also favourable. This is counter-intuitive as `electra`, a dynamically contextualized embedding, should be fitter, for it retains much more sense-specific knowledge as opposed to `char`. A possible reason is that `electra` training is a more difficult task, and it needs to undergo more training data than `sgns` and `char` to reach perfection. Finally, as noted in the previous section,



(a) en, sgns **(b)** en, char **(c)** en, electra

(d) fr, sgns **(e)** fr, char **(f)** fr, electra

(g) ru, sgns **(h)** ru, char **(i)** ru, electra

Figure 4.2: Visualization of gold and output embedding distributions across languages and embedding architectures.

the evaluation of the generated definitions is challenging in the setting we used.

4.6.2 Performances across languages

Metric scores (e.g. cosine similarity for embeddings or BLEU for texts) cannot be compared across different languages, so we refer to the overall team ranking from the shared task organizers (Mickus et al., 2022). Our system's behaviour is relatively strong and consistent when dealing with various languages, except for English. Assuming that the datasets are of similar quality, it is questionable to conclude that our model suits other languages more than English. Moreover, Figure 4.2 confirms that the English embeddings are not more peculiar than those of other languages.

We conjecture that some participants have focused on English (e.g. only submitting English systems and thus tuning hyperparameters on English), as it is a centred language in the research community. Instead, our data augmentation and architecture were designed to be language-agnostic, and we did not tune the hyperparameters for each specific language.

Feature	Category / Range	No. of instances
polysemy	Yes	65
	No	135
part-of-speech	Adj	56
	Adv	11
	Verb	37
	Noun	96
word frequency rank	67 – 11145	67
	11146 – 44416	66
	44417 – 905726	67
word length	3 – 5	85
	6 – 7	60
	8 – 17	55
definition length	1 – 6	71
	7 – 10	65
	11 – 39	64

Table 4.10: Statistics of the different subsets grouped by linguistic features.

4.6.3 Performances across linguistic features

We then look into the unified model's trial set predictions to interpret how scores vary across diverse linguistic annotations: polysemy, part of speech (POS), word length in characters, definition length in words, and word frequency. For categorical features, we group data by annotations; for numerical features, we divide the data into three subsets by percentiles: 0-33, 33-67, and 67-100. Statistics of the subsets are in Table 4.10. We list cosine similarity for reverse dictionary and lemma-BLEU for definition modelling.

A generic discovery is that the best scores of the two tracks emerge in differing subsets, regardless of the linguistic feature being analysed.

Polysemy	sgns		char		electra	
	cosine	lemma-BLEU	cosine	lemma-BLEU	cosine	lemma-BLEU
Yes	0.232	**4.34**	0.804	**3.20**	0.836	**3.61**
No	**0.360**	2.82	**0.813**	2.53	**0.845**	3.09

Table 4.11: Performances across polysemy annotations for `en`.

Polysemy Table 4.11 lists our unified system's results for words with either a single or multiple senses. It is slightly easier to achieve better cosine similarity for unambiguous words. Polysemous words have better BLEU, and `electra` has worse BLEU than `sgns`. This is counter-intuitive, as defining a polysemous word is harder, especially without context. As discussed, we hypothesize that this is due to the imperfect evaluation metric and `electra` embeddings being sub-optimal in quality.

POS	sgns		char		electra	
	cosine	lemma-BLEU	cosine	lemma-BLEU	cosine	lemma-BLEU
Adj	0.319	3.36	0.801	2.76	0.811	2.81
Adv	0.134	**6.56**	0.798	**5.45**	0.815	**5.93**
Verb	**0.383**	3.20	**0.839**	2.50	0.853	3.83
Noun	0.314	2.97	0.806	2.53	**0.860**	2.99

Table 4.12: Performance across POS tags for `en`.

Part of speech Next, numbers for the four POS tags that exist in the English trial set are laid out in Table 4.12. Strong cosine similarity is associated with verbs, although cosine numbers are close, except for adverbs. Adverbs, which have a small sample size, dominate high lemma-BLEU, probably because they are the least ambiguous.

Word length We then make three partitions by word lengths. Results in Table 4.13 suggest that shorter words have higher cosine, while longer words have slightly higher

Word length	sgns		char		electra	
	cosine	lemma-BLEU	cosine	lemma-BLEU	cosine	lemma-BLEU
short	**0.332**	3.19	**0.845**	2.58	0.817	3.10
medium	0.314	3.19	0.842	2.74	**0.867**	**3.41**
long	0.327	**3.66**	0.694	**3.00**	0.854	3.33

Table 4.13: Performances across word lengths for en.

lemma-BLEU. Numbers are closer for sgns and electra; we further investigate the grouping of char embeddings in Section 4.6.4.

Definition length	sgns		char		electra	
	cosine	lemma-BLEU	cosine	lemma-BLEU	cosine	lemma-BLEU
short	0.280	**4.51**	0.796	**3.60**	0.824	**4.89**
medium	0.318	3.48	0.814	2.73	0.848	2.76
long	**0.361**	1.83	**0.822**	1.80	**0.856**	1.93

Table 4.14: Performances across definition lengths for en.

Definition length Likewise in Table 4.14, we separate the trial data by the gold definition lengths. Much higher BLEU is seen when the model defines words linked with a shorter gold gloss, as generating a shorter sequence is easier. As we anticipate, when the model produces word embeddings for longer glosses, results are better too, potentially because more information can be encoded in a longer input.

Word frequency	sgns		char		electra	
	cosine	lemma-BLEU	cosine	lemma-BLEU	cosine	lemma-BLEU
low	0.250	3.53	0.805	**2.82**	0.850	3.30
medium	0.348	**3.54**	0.786	2.76	**0.864**	**3.38**
high	**0.357**	2.89	**0.839**	2.66	0.814	3.10

Table 4.15: Performances across word frequencies for en.

Word frequency Finally, Table 4.15 summarizes the results of the groups associated with low, medium, and high frequencies. From the results, we cannot establish an explicit trend across different task directions, embeddings, or word frequencies. This implies that the embedding quality and model performance might be word frequency-agnostic.

4.6.4 Observing the crescent with a telescope

After PCA retains the most distinguishing components, Figure 4.2 shows interesting patterns. Especially for `char`, we have observed clusters of points. We randomly label 25 English words and present them in Figure 4.3 and Figure 4.4, respectively for `char` and `electra`. We reveal that the clusters in `char`'s crescent are perfectly in tune with word lengths. For `electra`, more frequent words are closer to the origin. We do not notice a clear trend for `sgns`, for which a plot is included as Figure 4.5.

We can attribute the distinct patterns in different embeddings to their training paradigms: character-level word autoencoding for `char` and contextualized modelling for `electra`. This accounts for the largest cosine gap on `char` between long and short words seen earlier in Table 4.13 in the previous section. Intuitively, it is more difficult to train `char` autoencodings for longer words, so, in turn, embeddings for longer words possess inferior quality.

Within `char` embeddings, words are grouped by lengths, so future work might utilize this for word retrieval. Nonetheless, we are unsure of how the information of length or frequency aids sense-based tasks, like definition generation in our context.

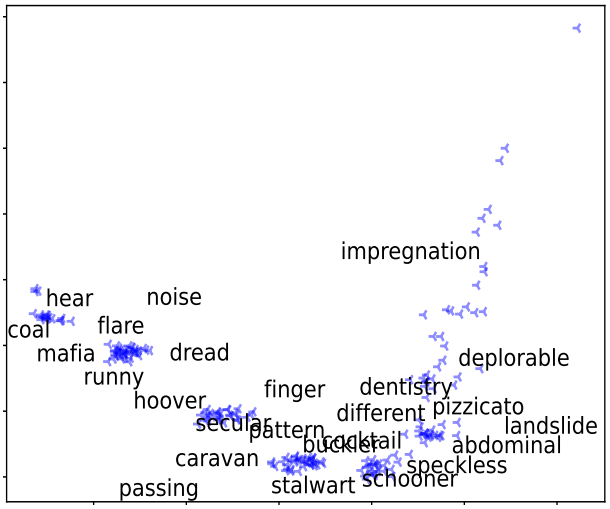


Figure 4.3: Gold English `char` embeddings with word labels.

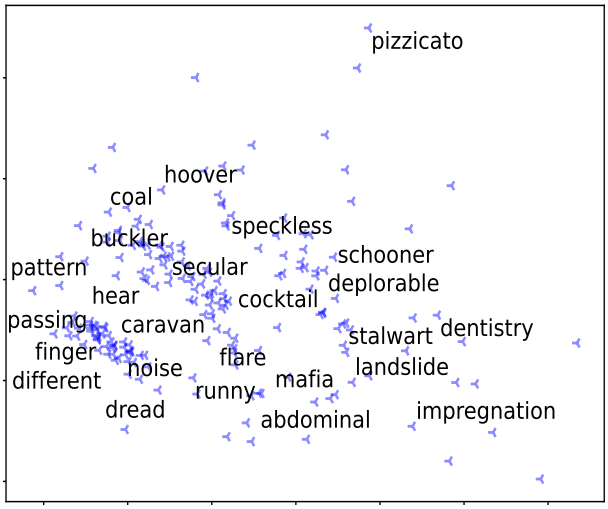


Figure 4.4: Gold English `electra` embeddings with word labels.

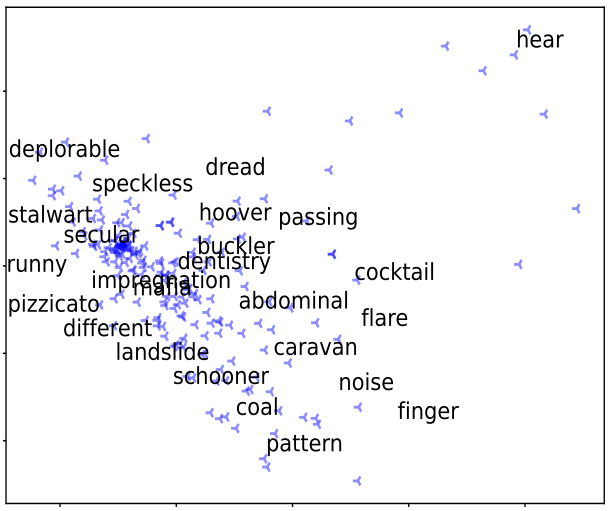


Figure 4.5: Gold English `sgns` embeddings with word labels.

4.7 Summary

This chapter investigated a self-data augmentation method for reverse dictionary and definition generation tasks. By assuming the source and target ends are parallel, we implemented a model similar to LASER with a shared representation space for both words and definitions. It uses a unified data and training strategy to train multiple generation directions and diverse objectives. We prove that our work outperforms the models separately trained on the individual tasks on conventional datasets. Our systems also achieved superior positions in a shared task at SemEval-2022 which benchmarked participating systems on lower-resourced datasets in five languages and three

embedding architectures. As an additional contribution to the field, we demonstrated the challenge of evaluating definitions in a low-resource situation by submitting an adversarial entry.

Our evaluation and analysis show that the proposed method trains better representations for words and is capable of generating definitions with higher quality indicated by human annotators. On low-resource data, we investigated the effect of different embedding methods, languages, and linguistic feature groups. We also revealed that embeddings trained with different algorithms display distinct patterns when projected to a lower-dimensional space.

Chapter 5

Data Augmentation for Code Generation Tasks

5.1 Introduction

Recent years have seen the rapid development of pre-trained models (PLMs) to enable knowledge transfer from generic texts to specific downstream tasks (Devlin et al., 2019; Liu et al., 2019). PLMs have been applied to the programming language domain as well, following the paradigm of continue-training text PLMs on code and text data and then fine-tuning them for specific tasks involving code (Kanade et al., 2020; Feng et al., 2020; Lu et al., 2021): similarity detection, error detection, completion, to name a few. Text PLMs are often adapted to programming languages by including code-specific modalities as part of the input like serialized syntax trees and data flows (the movement of values in the code) (Guo et al., 2021, 2022; Tipirneni et al., 2022). Such works have outperformed rule-based tools in various programming-related tasks, e.g. those in the CodeXGLUE benchmark (Lu et al., 2021). In terms of practical values, the advances facilitate applications ranging from companies' internal code management tools to public-facing programming assistants like Copilot and Codex, which greatly improve the coding experience for programmers.[10,11]

The abundance of raw code, which is usually achieved through harvesting code repositories, can be useful for pre-training regardless of a specific model architecture. However, programming data that meets certain downstream needs like code translation remains modest. We hypothesize that this is due to two factors: 1) code data are

[10] https://github.com/features/copilot
[11] https://openai.com/blog/openai-codex

still orders of magnitude smaller than texts publicly available; 2) manual code dataset curation requires the annotators to have good programming knowledge, which is a stricter requirement than employing human annotators to create text datasets. To offer a comparison in data sizes, code translation data in the CodeXGLUE benchmark is sized at 10K for training, whereas their natural language counterparts often include millions of training instances, for example, data offered at the yearly machine translation shared tasks (Kocmi et al., 2022).

While the research efforts have focused on pre-training code language models, not many prior works are known to mitigate data scarcity at the fine-tuning stage. In this work, we fill in the gap by enriching task-specific data for fine-tuning PLMs. To avoid expensive manual annotation costs, we explore two streams of automatic data augmentation methods that are suitable for fine-tuning any pre-trained code models. First, as generation-based augmentation, we adapt previous works in the field of machine translation, such as back-translation and monolingual copying to code generation tasks. As an analogy to multilingual machine translation, we also experiment with multilingual code summarization. In addition, this thesis proposes a novel numeric augmentation to improve number encoding and generation consistency, which is a token-level self-augmenting approach suitable specifically for code. We are keen to answer whether data augmentation is still relevant in the era of PLMs that adopt the pre-training fine-tuning paradigm, in the domain that deals with both programming and natural languages.

We experiment with the techniques on three typical generation tasks: 1) code translation, where a programming language is converted to another; 2) code summarization, where a textual description is produced from a block of code; 3) code synthesis where a code snippet should be generated from a natural language description together with classes and methods to use. Experiments show that even with limited resources at the fine-tuning stage, we can lift performance by 6.9% for code translation and 7.5% for code summarization compared to baselines. The augmentation approaches do not interfere with each other, and the best outcome is achieved when these are applied together. Through manual inspection and additional evaluation measures, we conclude that our methods lead to desirable enhancements special to code, namely better output code style and number correctness. However, when our data augmentation is applied to code synthesis, the results are less ideal; we summarize the possible drawbacks hoping to provide useful insights for future work.

In general, our study is the pioneering work in introducing automatic data augmen-

tation techniques to the fine-tuning of code language models. Since we demonstrate that substantial improvement can be achieved via data augmentation, this chapter poses a view supplementary to Feng et al. (2021)'s survey which considers data augmentation to have "minimal benefit for pre-trained models on in-domain data" for text classification. This work is based on the author's previous publication (Chen and Lampouras, 2023) with each person's contribution detailed in Section 1.3. Our code is expected to be made available publicly.[12]

5.2 Related Work

5.2.1 Preliminaries: pre-training and fine-tuning for code

Performing pre-training followed by fine-tuning is a popular paradigm in the field of NLP and has been adopted as a common approach to tasks involving code too (Lu et al., 2021). Recent research at the intersection of natural language and programming language processing concentrated on code model pre-training. Similar to the pre-training of text PLMs, code pre-training incorporates a large amount of programming language data (Kanade et al., 2020; Feng et al., 2020). Then the code PLMs can be fine-tuned for specific downstream tasks with task-specific data which is usually modest in size for programming languages. Specifically for generation tasks, while encoder-decoder and decoder-only models can be directly fine-tuned for generation, encoder-only models need to be appended with a decoder, which is often randomly initialized before fine-tuning. In the scope of our work, a complete code snippet or program is treated as a single line, where the line delimiters, if any, are considered to be tokens. Code data can be tokenized in the same way as text. Previous works optionally run a code tokenizer for code snippets followed by subword tokenization into a closed vocabulary.

5.2.2 Pre-trained code language models

Early attempts from Kanade et al. (2020) resulted in CuBERT, a model designed to generate contextualized embeddings for code. These embeddings function as a surrogate for code representation, enabling their application in a variety of classification tasks related to programming languages. Feng et al. (2020) built CodeBERT by further training RoBERTa (Liu et al., 2019) on bimodal text-code data with the replaced

[12] https://github.com/huawei-noah/noah-research/tree/master/NLP/DA4CodeGeneration

token detection objective introduced in ELECTRA (Clark et al., 2020). GraphCodeBERT takes a step further by incorporating extra training objectives that pay attention to code structure which is not common in texts, including data flow edge prediction and data-variable alignment (Guo et al., 2021).

Encoder PLMs are capable of capturing code embeddings for classification tasks, but they are not capable of generating programming languages. Hence apart from encoder PLMs, researchers have expanded uni-directional decoder-only models to the code domain too, e.g. Codex and Pangu-Coder (Chen et al., 2021; Christopoulou et al., 2022). Codex has been fine-tuned by first initializing from GPT models (Radford et al., 2019), whereas Pangu-Coder is trained from scratch in two stages involving both causal language modelling and masked language modelling.

Universal encoder-decoder code PLMs have also been presented: PyMT5, CodeT5, PLBART, UniXcoder, and StructCoder (Clement et al., 2020; Wang et al., 2021; Ahmad et al., 2021; Guo et al., 2022; Tipirneni et al., 2022). These models generally ingest more code data and leverage code structure-specific auxiliary objectives, which are the main differences from their counterpart text PLMs. Specifically, our experiments use UniXcoder (Guo et al., 2022) and StructCoder (Tipirneni et al., 2022) as they are the latest models claimed to attain top-notch performances on the downstream tasks we investigate. The former adopts attention masks to control encoder-decoder behaviours in a shared encoder-decoder network and incorporates flattened abstract syntax trees as input, with contrastive learning on code representations. The latter uses an encoder with separate attention matrices between any two of the code, syntax trees, and data flow, as well as a decoder that learns to predict the code, syntax trees and data flow graphs.

5.2.3 Data augmentation for code generation

Datasets for specific tasks concerning code are usually small, and data augmentation related to programming languages is largely unexplored. Early exploration combined cross-lingual masked modelling and iterative back-translation to build an unsupervised code transcompiler (Roziere et al., 2020). Despite not requiring translated code as training data, the model's performance is not on par with supervised training. Lately, Ahmad et al. (2023) ran code-to-text summarization followed by text-to-code generation, to obtain translation data between different programming languages using text summaries as a bridge. In contrast, one of our techniques is to train a text-to-code gen-

eration model in a back-translation style using the code-to-text summarization data. We note that our method is one-step while their approach takes two steps. While both approaches feature the concept of creating code data from other relevant tasks, our works differ in both the procedure and the intended end task.

On a broader scope, Yu et al. (2022) handcrafted rules for source code transformation; in comparison, we aim to investigate automatic methods which in no doubt is less costly. Also, we believe that techniques like dead code insertion and variable renaming inspired by malware obfuscation (You and Yim, 2010), as well as traditional text perturbation methods like string manipulation (e.g. token noising, swapping, deletion) can be useful. Nonetheless, these methods are task-agnostic, so we suggest that they could be more appropriate for the code pre-training stage rather than the task-specific fine-tuning stage.

5.3 Methodology

5.3.1 Generation-based augmentation

Back-translation for code translation Back-translation (Sennrich et al., 2016a) is a data augmentation technique that originated in machine translation, where an auxiliary model is used to construct pseudo-parallel data from monolingual resources. It can be straightforwardly applied to code translation. Formally, to train a code translation model $f()$ that converts a programming language PL_X into PL_Y, we first train an inverse model $g(PL_Y) \rightarrow PL_X$ with parallel data in PL_X-PL_Y pairs. Having the inverse model $g()$, extra monolingual data in PL_Y can be translated into $PL_{X'}$ to form pseudo-parallel code pairs $PL_{X'}$-PL_Y that can be used to train $f()$ in addition to the original PL_X-PL_Y data.

Multilingual back-translation for code summarization For code summarization, where a programming language (PL) is mapped into a natural language (NL), back-translation is not directly applicable. Since there hardly exist natural language summaries of code functions without code snippets in the first place, it is difficult to synthesize code data from natural language directly. Hence, we propose to use the summaries originally associated with a single programming language as a pivot for other programming languages. After inverting code-to-text data which has source side code available in multiple programming languages ($PL_1 \rightarrow NL_1, \ldots, PL_n \rightarrow NL_n$), we train

a multilingual text-to-code generator, which outputs a designated programming language given a natural language summary alongside language tag ($NL_k + tag_{\{1,\ldots n\}} \rightarrow \{PL_1, \ldots, PL_n\}$). This generator can iteratively produce code in different PLs by inputting text summaries regardless of the original $PL \rightarrow NL$ alignment. Effectively, while each natural language summary is originally paired with only a single programming language, it augments the data such that each summary now is paired with code from all programming languages. These backward-synthesized data, despite having a lower quality, can augment the training data multiple times in size for summarization.

5.3.2 Self-augmentation for code

Target copying for code translation Currey et al. (2017) suggested that adding monolingual data in the target language to learn an additional autoencoding (AE) objective benefits translation models trained on limited data. We migrate this objective to code translation by mixing the original parallel data $PL_X \rightarrow PL_Y$ and copied target autoencoding data $PL_Y \rightarrow PL_Y$. This almost compute-free approach effectively builds a multilingual encoder that enables knowledge transfer, given the high similarity between programming languages, namely the overlap of numerals, syntax tokens, reserved keywords, etc. This data augmentation process ensures that the decoder is only exposed to a single programming language PL_Y to not add complexity.

Multilingual code summarization In code summarization, as the target NL text is fundamentally divergent from the input PL, target copying (or an autoencoding objective) might not be useful. Instead, following the concept of leveraging multilingualism in encoders, we explore the feasibility of building a "multilingual" code summarization model $\{PL_1, \ldots, PL_n\} \rightarrow NL$. The system is designed for an arbitrary programming language and produces a natural language summary. Similar to the concept above, such a many-to-one model allows encoder knowledge sharing. This also exposes the decoder to more NL summaries which would be originally paired with other programming languages if trained monolingually.

5.3.3 Number-aware augmentation

Numeric self-augmentation Referenced variables and their values are unique components of programming languages. To enhance the representation of these values, previous works on pre-training suggested attending to appropriate modalities, e.g. data

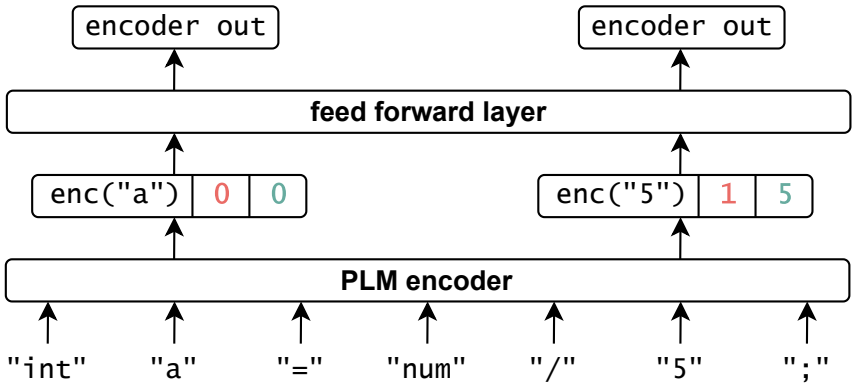


Figure 5.1: Numeric encoding with a PLM encoder, exemplifying how "a" is encoded as a string and "5" is encoded as a numeric value.

flow (Guo et al., 2021). However, to the best of our knowledge, there does not exist work that explores numerical features in the fine-tuning stage. This is probably because sophisticated handling of values might not be necessary for tasks like code translation or summarization since copying them over to the target side usually suffices. Yet, given a small data size at downstream fine-tuning, models might still only be exposed to sparse numerical input, and thus produce inconsistent outputs. Therefore we propose two simple techniques that aim to improve the numerical correctness in generation tasks involving code.

To improve model robustness, we first augment data by creating new instances where, in all code tokens containing a number, each digit is randomly replaced with another digit, consistently on both the source and target sides. We do not distinguish between purely numerical tokens and tokens including a number. For instance, a variable "num1" could become "num4" in the augmented code pair. The method guarantees that the number-swapped synthetic code is grammatical and compilable as long as the original code instance is.

Numeric encoding Apart from numerical data augmentation, we propose to include input numbers directly in the encoder output as mathematical values, complementary to their string embedding representations. This is backed by the motivation that numbers used in programming languages might not closely follow the distributional semantics theory that is behind the widespread use of word embeddings in the field of NLP: it is difficult to infer a number from surrounding tokens, especially when the number is large or customized.

As illustrated in Figure 5.1, we append two dimensions to the original encoder output. Particularly, one dimension (red, left) is a binary value (0/1) indicating whether the respective input is a number, while the other dimension (green, right) inherits the input's value, or 0 if the input is not numeric. The expanded embedding can be reduced to its original size via a feed-forward layer; such a change adds an extra layer but requires no modification to the pre-trained encoder itself.

5.4 Code Translation and Summarization Experiments

5.4.1 Tasks, datasets, and evaluation

We benchmark our methods on a code task suite named CodeXGLUE (Lu et al., 2021). Its translation task uses code originally developed in Java and then migrated to C#, so the corresponding C#-Java snippets are considered parallel. Training, validation, and test sizes are 10K, 0.5K, and 1K respectively. For back-translation, we translated 377K lines of monolingual Java, albeit out-of-domain from other CodeXGLUE tasks, into C#. To ensure that the target side consists of genuine data, we only experimented with the C#→Java direction as there is no other C# code in the benchmark dataset that can be used for back-translation.

Specifically for numerical augmentation, we experiment with the translation task because we are able to symmetrically change the numbers of both the source and target ends without affecting the parallelism. We take the training data and randomly substitute each digit with another one. We mix the number-substituted data with the original data to form a new training set which is of doubled size, but the validation and test sets remain the same.

The summarization task employs CodeSearchNet (Husain et al., 2020) and covers six languages: Ruby, JavaScript, Go, Python, Java, and PHP. Training sizes range from 25K to 250K, with a total of 908K; validation and test sets are between 1K and 15K for these languages. We performed multilingual back-translation by reversing the dataset so no external data is introduced; this leads to a five-fold BT data of 4.5M (908K×5). Consequently, all programming languages share an equal amount of original and synthetic data combined. Moreover, to compare the quality of neural back-translation against hand-written rule-based conversion, we created 80K JavaScript-summary pairs from Python-summary data by using `jsbuilder` to convert Python

to JavaScript (JS).[13]

We report code translation results in BLEU-4 (Papineni et al., 2002), exact line matches (EM, in %), and CodeBLEU (Ren et al., 2020).[14] Exact line match is the same as line-level accuracy with regard to the ground truth, which is a strict metric that considers both text-overlap and code functionality. CodeBLEU considers code structure in addition to BLEU; it is computed as a weighted sum of four accuracies: n-grams, weighted n-grams to feature reserved words in a programming language, code syntax, and data flow. Code summarization performance is measured by the de facto choice of BLEU-4 on text summaries. It is worth noting that when a generated code piece is evaluated against the reference, the metric computes each entire function as a single line rather than in separate lines.

5.4.2 Pre-trained code language models

For both code translation and summarization, we use the CodeXGLUE baseline, i.e. CodeBERT encoder with a randomly initialized Transformer decoder, as our baseline. We also use these inverse models for backwards-style data synthesis. We first fine-tune the PLMs on the augmented data, then continue fine-tuning on the original data. This is done for all experiments except for numerical augmentation where we mix the synthetic data with the training set, because we presume that the numerically augmented data should have no quality difference from the original data. Monolingual and multilingual summarization experiments also share the same training configuration. For numeric encoding with CodeBERT, we add a feed-forward layer after the encoder to make the baseline as deep as our proposed network to ensure that the comparison is fair. Specifically regarding whitespace used as part of the syntax by some programming languages, tab and newline characters are escaped and represented as strings (`\t` and `\n`) in the tokenized input. In this way, both horizontal and vertical syntax can be preserved.

To provide results with stronger baselines, we also test with GraphCodeBERT (Guo et al., 2021) for translation and UniXcoder (Guo et al., 2022) for summarization. These were the top-performing PLMs in their respective tasks when the work was carried out. We wish to verify the stability of our data augmentation performance across distinct PLM architectures.

[13] https://github.com/tvst/jsbuilder
[14] https://github.com/microsoft/CodeXGLUE

5.4.3 Technical configurations

Model and training details, with links to the preprocessing scripts, are summarized in Table 5.1. We retain the relevant PLMs' default configurations as much as possible, except for a grid search on the learning rate for code summarization with UniXcoder. We also changed the batch size to fully utilize our GPUs.

The randomly initialized Transformer decoder attached to CodeBERT and GraphCodeBERT has 6 layers, 12 heads, 768 hidden dimensions, and other hyperparameters as default in PyTorch. For the numeric encoding experiments with CodeBERT, we append 2 dimensions to CodeBERT's 768d encoder output, then transform it back to 768d using a linear layer. To ensure a fair comparison, a 768d-to-768d layer is also added to the baseline to make it as deep.

Hyperparameter	Value
PLM checkpoints	https://huggingface.co/microsoft/codebert-base
	https://huggingface.co/microsoft/graphcodebert-base
	https://huggingface.co/microsoft/unixcoder-base
	https://huggingface.co/microsoft/CodeGPT-small-java-adaptedGPT2
	https://github.com/reddy-lab-code-research/structcoder
trainable parameters	CodeBERT: 172.5M (with numeric encoding: + 591k)
	GraphCodeBERT: 172.5M
	UniXcoder: 126.5M
	CodeGPT: 124.4M
	StructCoder: 223.4M
learning rate	translation: $5e^{-5}$
	summarization: $1e^{-5}$, $5e^{-5}$, $\mathbf{1e^{-6}}$, $5e^{-6}$
	synthesis: $5e^{-5}$

Table 5.1: Model and training configurations.

All experiments are given a fixed budget of 100K updates. The training objective is cross-entropy, and we save the best checkpoint according to validation BLEU. At validation time, we use greedy decoding, but at test time, a beam size of 10 is used for inference. All results are based on a single run, but the experiments were benchmarked on PLMs of different architectures to reflect stability.

	BLEU	EM	CodeBLEU†
CodeBERT (Feng et al., 2020)			
reported	72.14	58.0	-
replicated	72.92	57.4	78.93 (72.92 / 73.61 / 87.08 / 82.10)
back-translation	77.34	61.4	83.36 (77.34 / 78.11 / **90.34** / 87.64)
+ autoencoding	**77.60**	**61.8**	**83.47** (**77.60** / **78.30** / 90.02 / **87.96**)
GraphCodeBERT (Guo et al., 2021)			
reported	72.64	58.8	-
replicated	72.66	58.9	78.55 (72.66 / 73.35 / 87.44 / 80.74)
back-translation	75.15	60.7	82.13 (75.15 / 75.86 / 90.06 / 87.46)
+ autoencoding	**76.15**	**62.5**	**82.88** (**76.15** / **76.87** / **90.54** / **87.95**)

†Average of four components (n-gram, weighted n-gram, syntax, and data flow accuracies).

Table 5.2: Test results for C#-to-Java translation.

	BLEU						
	Ruby	**JS**	**Go**	**Python**	**Java**	**PHP**	***Average***
CodeBERT (Feng et al., 2020)							
monolingual, reported	12.16	14.90	18.07	19.06	17.65	25.16	17.83
monolingual, replicated	12.39	14.13	17.89	18.22	18.66	25.14	17.73
+ rule-based translation	-	15.35	-	-	-	-	-
+ back-translation	13.76	15.00	18.30	18.60	19.64	25.69	18.50
multilingual	**14.93**	15.53	18.68	18.71	19.70	25.96	18.92
+ rule-based translation	14.58	15.65	18.77	18.95	**19.86**	25.98	18.97
+ back-translation	14.91	**15.81**	**18.88**	**18.97**	19.69	**26.10**	**19.06**
UniXcoder (Guo et al., 2022)							
monolingual, reported	14.87	15.85	19.07	19.13	20.31	26.54	19.30
monolingual, replicated	14.81	15.28	18.93	19.05	20.22	26.66	19.16
multilingual	**15.15**	15.64	19.03	19.22	**20.45**	26.59	19.35
+ back-translation	14.94	**15.85**	**19.29**	**19.36**	20.43	**26.69**	**19.43**

Table 5.3: Test results for code summarization.

5.4.4 Results and discussions

Generation-based and self-augmentation We first show translation results in Table 5.2, where our replicated baseline has similar results to those reported in previous works. Back-translation augmentation surpasses baselines by a large margin for both PLMs; on top of it, autoencoding brings a small gain. The observations are the same for both CodeBERT and GraphCodeBERT, indicating that our data augmentation methods achieve consistent improvements across different PLM architectures. The breakdown of the CodeBLEU metric shows that all four components benefit from data augmentation.

We then report summarization results in Table 5.3. We have two findings: 1) our proposal of a single multilingual summarization model is better than training separate monolingual models as we replicated from previous research. 2) Back-translation steadily improves performance in addition to multilingual training. For CodeBERT, rule-based and neural back-translation produces close results. An interesting pattern from both PLMs is that back-translation helps Ruby and Java less than other programming languages. Furthermore, training a single multilingual model achieves better performance and is associated with smaller storage requirements. Potentially, this is due to transfer learning between programming languages and the increase in natural language data size on the output side. Finally, we observe a smaller improvement on UniXcoder compared to CodeBERT. This could be due to the already strong performance of the baseline.

Number-aware augmentation Results of our proposed augmentation involving numbers are listed in Table 5.4. Given that the methods are designed to focus on number consistency, we compute and append output token-level accuracies to Table 5.4, with a distinction between numeric and non-numeric tokens. The upper half of the table shows improved scores for number shuffling in all metrics, even seeing a small improvement for non-numeric accuracy.

On the other hand, our numeric encoding underperforms the baseline. A potential reason is that directly appending an input number that is orders of magnitude larger than a vector of weights leads to instability. To accommodate this discrepancy, we investigate linear and logarithmic scaling of the encoded values. As the scaling factor gets smaller, we find that the metric scores gradually catch up. The optimal is a logarithmic transformation, whereby the model attains the highest performance.

	BLEU	EM	CodeBLEU	Token Accuracy	
				num.	non-num.
CodeBERT					
replicated + FFN	72.88	58.0	78.07	74.50	86.72
+ numeric augmentation	**74.00**	**59.5**	**79.43**	**76.14**	**87.30**
numeric encoding	72.95	58.1	78.77	73.74	86.84
+ numeric augmentation with value scaling					
$\times 10^2$	71.32	51.6	77.71	72.48	85.98
$\times 1$ (no scaling)	72.51	57.4	78.45	72.92	86.49
$\times 10^{-2}$	73.48	**59.2**	79.41	74.11	87.11
$\times 10^{-4}$	74.01	58.9	79.73	74.93	**87.48**
$\log_{10}()$	**74.16**	59.1	**79.84**	**75.22**	87.32

Table 5.4: Test results for C#→Java translation with number augmentation and encoding.

	BLEU	EM	CodeBLEU	Token Accuracy	
				num.	non-num.
CodeBERT					
replicated	72.92	57.4	78.93	74.64	87.54
back-translation	77.34	**61.4**	83.36	78.09	88.62
+ num. aug. on BT and original	77.37	60.9	83.43	77.16	88.55
+ num. aug. on original	**77.69**	61.0	**83.44**	**78.54**	**88.69**
back-translation + autoencoding	77.60	61.8	83.47	77.16	88.64
+ num. aug. on original	**77.96**	**62.0**	**83.63**	78.01	**88.79**

Table 5.5: Test results for C#→Java translation with multiple augmentation techniques stacked togerther.

We conclude that the numerical approaches aid number generation without compromising non-numbers, and the improvement in number correctness is generally consistent with the improvement in BLEU and EM. Additional visualization shown later in Section 5.5.2 implies that our numeral-augmented models can maintain numeric consistency even when the output is extremely long and complicated.

Finally, we examine if our proposed methods, namely back-translation, autoencoding, and numeric augmentation, can work orthogonally. Table 5.5 shows that better results are achieved when numeric augmentation is applied to the original data, but not to the back-translated data. This is probably because back-translations are already of inferior quality to original data, e.g. parallelism might have been violated, so numerical augmentation introduces extra noise. Nevertheless, combining back-translation and autoencoding with numeric augmentation over the original data leads to the best outcome when evaluated by all metrics.

5.5 Manual Analysis

We conduct manual inspections on the code generated in the translation task. We have two observations from a qualitative aspect: 1) the data-augmented model better follows the coding convention used in the target language, exemplified by how element retrieval is implemented; 2) we find improved number accuracy and consistency in the number-augmented models. Finally, we point out some imperfections in the gold translation output, but we leave in-depth investigation and refinement for future work.

5.5.1 Element retrieval methods

Upon inspecting the translation test outputs, we find that our data-augmented model—following Java programming conventions instead of following the input code style—has been reasonably exposed to the target language. We present test instances focused on element retrieval methods, by listing sources, references, and outputs from the CodeBERT baseline and our BT-augmented model in Table 5.6. The baseline tends to retrieve an element through reference to its index, which is generically correct. Nonetheless, the data-augmented model closely follows the Java coding convention in an object-oriented way where the inbuilt method `get()` is favoured over directly accessing the attributes by indices. This highlights the importance of feeding more programming data naturally written in the target language to the model, which is a key

```
// test #85
C# source      ... GetEscherRecord(int index){return escherRecords[index];}
Java reference ... getEscherRecord(int index){return escherRecords.get(index);}
baseline       ... getEscherRecord(int index) {return escherRecords[index];}
DA model       ... getEscherRecord(int index) {return escherRecords.get(index);}

// test #90
C# source      public virtual IQueryNode GetChild(){return GetChildren()[0];}
Java reference public QueryNode getChild() {return getChildren().get(0);}
baseline       public QueryNode getChild() {return getChildren() == 0);}
DA model       public QueryNode getChild() {return getChildren().get(0);}

// test #978
C# source      public virtual SrndQuery GetSubQuery(int qn) { return m_queries[qn]; }
Java reference public SrndQuery getSubQuery(int qn) {return queries.get(qn);}
baseline       public SrndQuery getSubQuery(int qn) {return queries[qn];}
DA model       public SrndQuery getSubQuery(int qn) {return queries.get(qn); }
```

Table 5.6: C#-Java output translations of *element retrieval methods*, before and after data augmentation.

part of our generation-based methods.

5.5.2 Numeric consistency

Complementing the number accuracy figures reported in Section 5.4.4, we list translation outputs containing numbers in Table 5.7 for visualization. Despite that in code translation, most numbers simply need to be copied over, the baseline model still makes mistakes in those cases. The examples convey the idea that our numeric augmentation models can improve number consistency even in very long and complicated outputs. In the baseline outputs, for example in test #436, number incorrectness further leads to undesirable hallucinations, which can be prevented in the number-augmented model's output. Test # 716 represents a complex for-loop execution involving numbers, which has been handled correctly by the model trained on augmented data too.

5.5.3 Test set imperfection

Finally, we would like to note that in the automatically constructed translation test set, a small proportion of code pairs seem not to be perfectly parallel. The divergence can lead to an inaccurate estimate of translation output performance. We record a few examples of these imperfections in Table 5.8. In these cases, the code in two languages will not function exactly the same when being executed. In code generation, correct model outputs could be penalized in terms of text metrics when scored against those instances, thus undermining the evaluation setup.

```
// test #131
C# source       public ScaleClusterRequest(): base("CS", "2015-12-15", "ScaleCluster"
                    , "cs", "openAPI"){UriPattern = "/clusters/[ClusterId]";
                    Method = MethodType.PUT;}
Java reference  public ScaleClusterRequest() {super("CS", "2015-12-15", "ScaleCluster"
                    , "csk");setUriPattern("/clusters/[ClusterId]");
                    setMethod(MethodType.PUT);}
baseline        publicscaleClusterRequest() {super("CS", "2018-12-15", "ScaleCluster"
                    , "cs");setUriPattern("/clusters/[ClusterId]");
                    setMethod(MethodType.PUT);}
DA model        public ClusterRequest() {super("CS", "2015-12-15", "ScaleCluster"
                    , "cs");setUriPattern("/clusters/[ClusterId]");
                    setMethod(MethodType.PUT);}

// test #436
C# source       public void CopyTo(byte[] b, int o){FormatHexByte(b, o + 0, w1);
                    FormatHexByte(b, o + 8, w2);FormatHexByte(b, o + 16, w3);
                    FormatHexByte(b, o + 24, w4);FormatHexByte(b, o + 32, w5);}
Java reference  public void copyTo(byte[] b, int o) {formatHexByte(b, o + 0, w1);
                    formatHexByte(b, o + 8, w2);formatHexByte(b, o + 16, w3);
                    formatHexByte(b, o + 24, w4);formatHexByte(b, o + 32, w5);}
baseline        public void copyTo(byte[] b, int o) {formatHexByte(b, o1);
                    formatHexByte(b, o2);formatHexByte(b, o2);
                    formatHexByte(b, o3);formatHexByte(b,o + 24, w4);
                    formatHexByte(b, o + 32, w5);}
DA model        public void copyTo(int[] b, int o) {formatHexByte(b, o + 0, w1);
                    formatHexByte(b, o + 8, w2);formatHexByte(b, o + 16, w3);
                    formatHexByte(b, o + 24, w4);formatHexByte(b, o + 32, w5);}

// test #716
C# source       public override void Decode(byte[] blocks, int blocksOffset, int[]
                    values, int valuesOffset, int iterations){for (int j = 0;
                    j < iterations; ++j){var block = blocks[blocksOffset++];
                    values[valuesOffset++] = ((int)((uint)block >> 7)) & 1;
                    values[valuesOffset++] = ((int)((uint)block >> 6)) & 1;
                    values[valuesOffset++] = ((int)((uint)block >> 5)) & 1;
                    values[valuesOffset++] = ((int)((uint)block >> 4)) & 1;
                    values[valuesOffset++] = ((int)((uint)block >> 3)) & 1;
                    values[valuesOffset++] = ((int)((uint)block >> 2)) & 1;
                    values[valuesOffset++] = ((int)((uint)block >> 1)) & 1;
                    values[valuesOffset++] = block & 1;}}
Java reference  public void decode(byte[] blocks, int blocksOffset, int[]
                    values, int valuesOffset, int iterations) {for (int j = 0;
                    j < iterations; ++j) {final byte block = blocks[blocksOffset++];
                    values[valuesOffset++] = (block >>> 7) & 1;
                    values[valuesOffset++] = (block >>> 6) & 1;
                    values[valuesOffset++] = (block >>> 5) & 1;
                    values[valuesOffset++] = (block >>> 4) & 1;
                    values[valuesOffset++] = (block >>> 3) & 1;
                    values[valuesOffset++] = (block >>> 2) & 1;
                    values[valuesOffset++] = (block >>> 1) & 1;
                    values[valuesOffset++] = block & 1;}}
baseline        public void decode(byte[] blocks, int blocksOffset, int[]
                    values, int valuesOffset, int iterations) {for (int j = 0;
                    j < iterations; ++j) {final byte block = blocks[blocksOffset++];
                    values[valuesOffset++] = (block >>> 7) & 1;
                    values[valuesOffset++] = (block >>> 6) & 1;
                    values[valuesOffset++] = (block >>> 5) & 1;
                    values[valuesOffset++] = (block >>> 4) & 1;
                    values[valuesOffset++] = (block >>> 4) & 1;
                    values[valuesOffset++] = (block >>> 2) & 1;
                    values[valuesOffset++] = (block >>> 1) & 1;
                    values[valuesOffset++] = block & 1;}}
DA model        public void decode(byte[] blocks, int blocksOffset, int[]
                    values, int valuesOffset, int iterations) {for (int j = 0;
                    j < iterations; ++j) {final byte block = blocks[blocksOffset++];
                    values[valuesOffset++] = (block >>> 7) & 1;
                    values[valuesOffset++] = (block >>> 6) & 1;
                    values[valuesOffset++] = (block >>> 5) & 1;
                    values[valuesOffset++] = (block >>> 4) & 1;
                    values[valuesOffset++] = (block >>> 3) & 1;
                    values[valuesOffset++] = (block >>> 2) & 1;
                    values[valuesOffset++] = (block >>> 1) & 1;
                    values[valuesOffset++] = block & 1;}}
```

Table 5.7: C#-Java output translations containing *numbers*, before and after number-aware augmentation.

```
// test #307
C# source      public override string ToString(){return "IndexSearcher("
                  + reader + "; executor=" + executor + ")";}
Java reference public String toString() {return "IndexSearcher("
                  + reader + "; executor=" + executor
                  + "; sliceExecutionControlPlane " + sliceExecutor + ")";}

// test #518
C# source      public override PushConnection OpenPush() throw
                  {new NGit.Errors.NotSupportedException(
                     JGitText.Get().pushIsNotSupportedForBundleTransport);}
Java reference public PushConnection openPush() throws
                  {TransportException return new TcpPushConnection();}

// test #892
C# source      public Builder(): base(){lastDocID = -1;wordNum = -1;word = 0;}
Java reference public Builder() {this(true);}

// test #902
C# source      public override string ToString(){return "term="+ term+", field="
                   +field+", value="+value;}
Java reference public String toString() {return "term="+term+", field="
                   +field+", value="+valueToString()+",docIDUpto="+docIDUpto;}
```

Table 5.8: C#-Java test instances that are not perfectly parallel, with divergence shown in bold.

5.6 Insights from Code Synthesis

In addition to code translation and summarization, we also attempted data augmentation for a code synthesis task from CodeXGLUE's benchmark. The task samples data from CONCODE (Iyer et al., 2018) at 100K, 2K, and 2K for training, validation, and testing correspondingly. The source side contains a text description of the required functionality, as well as the available class variable and function names that can be used. The target is the corresponding Java code.

For code synthesis, while reversing the summarization data is a natural solution to augment training, the difficulty lies in forming the class environment (usable variables and methods visible to the model). We parse the code in a summarization instance to obtain the positive tokens that are used to form the correct code, as well as randomly sample tokens from other genuine code data as negative signals. In other words, from $PL \rightarrow NL$ pairs, we construct code synthesis data $NL + parse(PL) + sample(PL') \rightarrow PL$. This method is able to leverage existing summarization data to train code synthesis from text with a code parser.

Technically, we reverse the Java summary data in CodeXGLUE to create 181K synthetic data. In detail, to get available variable and method names, the code is parsed

by `javalang` into tokens.[15] Following CodeXGLUE, we use CodeGPT, which is a model based on GPT-2 but adapted to Java (Lu et al., 2021), as our baseline model. We further experiment with StructCoder (Tipirneni et al., 2022) which is a more up-to-date, and probably state-of-the-art pre-trained code language model when the work is carried out.

	BLEU	**EM**	**CodeBLEU**
CodeGPT on validation			
replicate	28.13	16.1	31.65
+ augmentation	**29.04**	**16.6**	**32.35**
StructCoder on validation			
replicate	37.30	18.2	40.42
+ augmentation	**37.48**	**18.7**	**40.47**
CodeGPT on test			
paper	32.79	20.1	35.98
replicate	32.66	**20.1**	35.89
+ augmentation	**33.45**	19.2	**36.47**
StructCoder on test			
paper	40.91	22.4	44.77
replicate	**41.57**	**22.6**	**44.61**
+ augmentation	41.32	21.4	44.04

Table 5.9: Results for code synthesis.

We report our experimental results in 5.9. Similar to code translation evaluation, the code generations are scored by three metrics: BLEU, EM, and CodeBLEU with respect to the gold output label. Note that we report the validation set performance in addition to the test. This is because the test references are not publicly available and the test predictions need to be sent to the CodeXGLUE authors for evaluation. Reporting the validation numbers could aid reproducibility and comparison. By looking at the result table, we notice that our augmentation work yields a small gain consistently on validation and test sets for CodeGPT. However, it does not improve upon the latest PLM with a small margin behind.

We conjecture a few possible reasons. First of all, we argue that code synthesis is

[15] https://github.com/c2nes/javalang

fundamentally a more challenging task than code translation and summarization. The information required to generate a translation or a summary is already present in the input—the input contains more (summarization), or at least equal (translation) information compared to the output end; in contrast, code synthesis output is more open-ended and variable. Model-wise, StructCoder is remarkably stronger than CodeGPT, thus the room for improvement may be small for data augmentation. Finally, the summarization data we used to augment the synthesis task could be different in terms of topic, style, etc, resulting in a domain drift. Gathering larger and in-domain augmented data might be beneficial. We leave the investigation of these hypotheses to future work.

5.7 Discussions on Code Generation Metrics

Our evaluation and findings rely on multiple string-based metrics: BLEU, exact line match, CodeBLEU, and token-level accuracy. However, in works where code generation from text instructions is concerned, test case pass rate (e.g. pass@K, where K is the number of hypothesis generations) is widely considered (Chen et al., 2021). Given the similar advantage that both types of metrics can be run quickly and automatically, we discuss the pros and cons of these metrics in this section.

Both text-based metrics and the test pass rate can reflect code quality. The test pass rate directly evaluates whether the functionality of the generated code matches certain requirements reflected through many test cases. It does not require a reference code but needs a comprehensive set of test cases. It is not affected by code readability. On the contrary, string-based metrics reflect the degree of overlap between the generated code and the reference code. Therefore, these metrics can suggest code readability, which is very important if the code needs to be maintained by a programmer. Although not fully explicit, some of those metrics also consider functionality or code structure, e.g. exact line match and CodeBLEU, as we explained in Section 5.4.1. Nonetheless, erroneous code will not be heavily penalized.

In general, we argue that the unit test case pass rate is more suitable for cases where the code is deployed after generation without human intervention, while string-based metrics could be more favourable if human programmers need to read and modify the generated code. In this situation, code that cannot be compiled or executed immediately is not a severe problem because it can be easily fixed with the help of coding tools in modern development environments. Thus, we argue that our choice of evaluation

metrics is reasonable, but as a future direction, it is sensible to combine the two types of metrics to evaluate code generation models.

5.8 Summary

In this chapter, we explored a few data augmentation approaches: back-translation, multilingualism, as well as numeric enhancement, and adapted them to the fine-tuning of pre-trained models for programming languages. The application of such is relatively new in the field. We answered the two questions we raised earlier: Our low-cost augmentation significantly improves metric scores for code translation and code summarization on top of various PLMs: CodeBERT, GraphCodeBERT, and UniXcoder. Results are less ideal for code synthesis, so we provided some insights hoping to serve future studies. Manual inspection captured qualitative improvements in element retrieval methods and number correctness in the translated code. We in addition offered our view on the role of traditional text metrics for evaluating code.

Our contribution focused on fine-tuning the smaller-scale PLMs compared to today's large language models (LLMs), which enjoy the emergent capabilities brought by billions of parameters and massive pre-training data. Nonetheless, it has been shown that fine-tuning still helps foundation LLMs to generalize to downstream tasks (Sanh et al., 2022). During the LLM fine-tuning phase, our work can be applicable since supervised code data are rather modest in size. Specifically, our multilingual and translation methods can contribute to research on multilingual and multi-task instructions based on machine translation (Muennighoff et al., 2023). Last but not least, our methods when applied to PLMs at the size of a few million parameters are still useful to the community from a deployable and sustainable perspective.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

We have investigated automatic data augmentation methods for language generation which faces data scarcity in many situations. Overall, several techniques inspired by the widely-researched neural machine translation area were explored. The thesis started with a novel proposal of automatic parallel sentence retrieval by constraining translation decoding with a trie. Then we applied multi-task learning with augmented data for reverse dictionary and definition generation. Finally, we investigated whether machine translation data augmentation approaches can be applied to code generation tasks on top of pre-trained language models. As detailed below, our research efforts and findings can be summarized from three aspects: novelty, data augmentation performance, and evaluation.

New research ideas studied This thesis continually made efforts to introduce novel ideas to the research community. Chapter 3 suggested the use of a neural translation system itself to identify more parallel data via trie-constrained decoding. In Chapter 4, we proposed a unified architecture to learn both reverse dictionary and definition generation tasks simultaneously with self-data augmentation. Many techniques used in Chapter 5 were inspired by machine translation; nonetheless, we devised a way to encode number inputs as numeric values as opposed to strings as traditionally done in natural language processing.

Data augmentation performance Chapter 3 presented a relatively self-contained method to automatically mine for parallel sentences for machine translation. It can

retrieve a reasonable amount of sentences compared to recent works, but it underperforms multilingual sentence representations derived from translation models. Chapter 4's self-data augmentation for unified representation learning achieved superior performance for both reverse dictionary and definition generation, across datasets with different languages, embeddings, and data availability. We also won many tracks in a shared task at an established international workshop. Finally, we demonstrate in Chapter 5 that both self-data augmentation and generation-based data augmentation inspired by machine translation can bring substantial enhancements to code translation and summarization. Our results also proved that it is beneficial to perform data augmentation for fine-tuning pre-trained language models.

Evaluation and metrics Empirical studies use test set scores from automatic metrics to benchmark a system's performance. The contribution becomes less verifiable if there is a defect in the test set or the metric. Our work in the thesis is no different, but we have demonstrated rigour in test set inspection and reflection on the metrics adopted throughout, which we recommend to all practitioners. Chapter 3 reported the train-test overlap in a common benchmark used by multiple parallel sentence mining works and we stayed clear of the contaminated data. In Chapter 4, we acknowledged the difficulty in definition generation evaluation using common text generation metrics, so we conducted a human evaluation to show our model's performance. We additionally designed an adversarial submission to a shared task to signal this to the wider community. Finally in Chapter 5 we found imperfections in a small test subset for code translation. We also justified our choice of metrics by discussing the pros and cons of string-based and code function-based evaluation.

6.2 Future Work

Large language models (LLMs) have started playing a pivotal role in natural language processing as the thesis is being carried out. Although during pre-training these general-purpose models have ingested much more data than specific-purpose models, they need to be prompted or instruction-tuned to interact with users and perform specific tasks. An intriguing question revolves around understanding the function of data in LLM fine-tuning—whether it introduces new knowledge or merely alters the response format. Presently, there is no definitive answer as the varying sizes of the instruction datasets hint at both possibilities (Wei et al., 2022; Zhou et al., 2023). A

data and model size-controlled analysis is needed to derive a definitive answer. Whilst data augmentation has traditionally been associated with benefits like increased robustness, regularization, and external knowledge incorporation, its role might change in the LLM era.

Furthermore, diverse and high-quality instruction data may still necessitate human annotation (Mishra et al., 2022; Zhou et al., 2023). Recent workarounds include using a model to generate data for itself (Wang et al., 2023; Li et al., 2023b) and using machine translation to support lower-resourced languages (Li et al., 2023a; Chen et al., 2024). We suggest that a possible research direction is to automatically retrieve valid question-answer pairs from the extensively available raw data on the web, similar to parallel sentence mining works.

Bibliography

Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified pre-training for program understanding and generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2655–2668, Online. Association for Computational Linguistics.

Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2023. Summarize and generate to back-translate: Unsupervised translation of programming languages. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 1528–1542, Dubrovnik, Croatia. Association for Computational Linguistics.

Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. 2017. Guided open vocabulary image captioning with constrained beam search. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 936–945, Copenhagen, Denmark. Association for Computational Linguistics.

Mikel Artetxe and Holger Schwenk. 2019a. Margin-based parallel corpus mining with multilingual sentence embeddings. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3197–3203, Florence, Italy. Association for Computational Linguistics.

Mikel Artetxe and Holger Schwenk. 2019b. Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond. *Transactions of the Association for Computational Linguistics*, 7:597–610.

Andoni Azpeitia, Thierry Etchegoyhen, and Eva Martínez Garcia. 2018. Extracting parallel sentences from comparable corpora with STACC variants. In *11th Workshop on Building and Using Comparable Corpora*, Paris, France. European Language Resources Association (ELRA).

Andoni Azpeitia, Thierry Etchegoyhen, and Eva Martínez Garcia. 2017. Weighted set-theoretic alignment of comparable sentences. In *Proceedings of the 10th Workshop on Building and Using Comparable Corpora*, pages 41–45, Vancouver, Canada. Association for Computational Linguistics.

Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. In *NeurIPS Deep Learning Symposium*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, Conference Track*.

Marta Bañón, Pinzhen Chen, Barry Haddow, Kenneth Heafield, Hieu Hoang, Miquel Esplà-Gomis, Mikel L. Forcada, Amir Kamran, Faheem Kirefu, Philipp Koehn, Sergio Ortiz Rojas, Leopoldo Pla Sempere, Gema Ramírez-Sánchez, Elsa Sarrías, Marek Strelec, Brian Thompson, William Waites, Dion Wiggins, and Jaume Zaragoza. 2020. ParaCrawl: Web-scale acquisition of parallel corpora. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4555–4567, Online. Association for Computational Linguistics.

Michele Bevilacqua, Marco Maru, and Roberto Navigli. 2020. Generationary or "how we went beyond word sense inventories and learned to gloss". In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7207–7221, Online. Association for Computational Linguistics.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Barry Haddow, Matthias Huck, Chris Hokamp, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Carolina Scarton, Lucia Specia, and Marco Turchi. 2015. Findings of the 2015 workshop on statistical machine translation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 1–46, Lisbon, Portugal. Association for Computational Linguistics.

Ondřej Bojar, Rudolf Rosa, and Aleš Tamchyna. 2013. Chimera – three heads for English-to-Czech translation. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 92–98, Sofia, Bulgaria. Association for Computational Linguistics.

Tom Bosc and Pascal Vincent. 2018. Auto-encoding dictionary definitions into consistent word embeddings. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1522–1532, Brussels, Belgium. Association for Computational Linguistics.

Houda Bouamor and Hassan Sajjad. 2018. H2@BUCC18: Parallel sentence extraction from comparable corpora using multilingual sentence embeddings. In *11th Workshop on Building and Using Comparable Corpora*, Paris, France. European Language Resources Association (ELRA).

Peter F. Brown, Jennifer C. Lai, and Robert L. Mercer. 1991. Aligning sentences in parallel corpora. In *29th Annual Meeting of the Association for Computational Linguistics*, pages 169–176, Berkeley, California, USA. Association for Computational Linguistics.

Ting-Yun Chang and Yun-Nung Chen. 2019. What does this word mean? explaining contextualized embeddings with natural language definition. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6064–6070, Hong Kong, China. Association for Computational Linguistics.

Ting-Yun Chang, Ta-Chung Chi, Shang-Chi Tsai, and Yun-Nung Chen. 2018. xSense: Learning sense-separated sparse representations and textual definitions for explainable word sense networks. *arXiv preprint*.

Boxing Chen and Colin Cherry. 2014. A systematic comparison of smoothing techniques for sentence-level BLEU. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 362–367, Baltimore, Maryland, USA. Association for Computational Linguistics.

Jiaao Chen, Derek Tam, Colin Raffel, Mohit Bansal, and Diyi Yang. 2023. An empirical survey of data augmentation for limited data learning in NLP. *Transactions of the Association for Computational Linguistics*, 11:191–211.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail

Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *arXiv preprint*.

Pinzhen Chen, Nikolay Bogoychev, and Ulrich Germann. 2020a. Character mapping and ad-hoc adaptation: Edinburgh's IWSLT 2020 open domain translation system. In *Proceedings of the 17th International Conference on Spoken Language Translation*, pages 122–129, Online. Association for Computational Linguistics.

Pinzhen Chen, Nikolay Bogoychev, Kenneth Heafield, and Faheem Kirefu. 2020b. Parallel sentence mining by constrained decoding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1672–1678, Online. Association for Computational Linguistics.

Pinzhen Chen and Kenneth Heafield. 2022. Approaching neural Chinese word segmentation as a low-resource machine translation task. In *Proceedings of the 36th Pacific Asia Conference on Language, Information and Computation*, pages 600–606, Manila, Philippines. Association for Computational Linguistics.

Pinzhen Chen, Shaoxiong Ji, Nikolay Bogoychev, Andrey Kutuzov, Barry Haddow, and Kenneth Heafield. 2024. Monolingual or multilingual instruction tuning: Which makes a better Alpaca. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 1347–1356, St. Julian's, Malta. Association for Computational Linguistics.

Pinzhen Chen and Gerasimos Lampouras. 2023. Exploring data augmentation for code generation tasks. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 1542–1550, Dubrovnik, Croatia. Association for Computational Linguistics.

Pinzhen Chen and Zheng Zhao. 2022a. Edinburgh at SemEval-2022 task 1: Jointly fishing for word embeddings and definitions. In *Proceedings of the 16th Interna-*

tional Workshop on Semantic Evaluation (SemEval-2022), pages 75–81, Seattle, United States. Association for Computational Linguistics.

Pinzhen Chen and Zheng Zhao. 2022b. A unified model for reverse dictionary and definition modelling. In *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 8–13, Online only. Association for Computational Linguistics.

Yong Cheng, Lu Jiang, Wolfgang Macherey, and Jacob Eisenstein. 2020. AdvAug: Robust adversarial augmentation for neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5961–5970, Online. Association for Computational Linguistics.

Fenia Christopoulou, Gerasimos Lampouras, Milan Gritta, Guchun Zhang, Yinpeng Guo, Zhongqi Li, Qi Zhang, Meng Xiao, Bo Shen, Lin Li, Hao Yu, Li Yan, Pingyi Zhou, Xin Wang, Yuchi Ma, Ignacio Iacobacci, Yasheng Wang, Guangtai Liang, Jiansheng Wei, Xin Jiang, Qianxiang Wang, and Qun Liu. 2022. Pangu-coder: Program synthesis with function-level language modeling. *arXiv preprint*.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*.

Colin Clement, Dawn Drain, Jonathan Timcheck, Alexey Svyatkovskiy, and Neel Sundaresan. 2020. PyMT5: multi-mode translation of natural language and python code with transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9052–9065, Online. Association for Computational Linguistics.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Anna Currey, Antonio Valerio Miceli Barone, and Kenneth Heafield. 2017. Copied monolingual data improves low-resource neural machine translation. In *Proceedings*

of the Second Conference on Machine Translation, pages 148–156, Copenhagen, Denmark. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. 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 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Cristina España-Bonet, Ádám Csaba Varga, Alberto Barrón-Cedeño, and Josef van Genabith. 2017. An empirical analysis of nmt-derived interlingual embeddings and their use in parallel sentence identification. *IEEE Journal of Selected Topics in Signal Processing*, 11(8):1340–1350.

Thierry Etchegoyhen and Andoni Azpeitia. 2016. Set-theoretic alignment for comparable corpora. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2009–2018, Berlin, Germany. Association for Computational Linguistics.

Marzieh Fadaee, Arianna Bisazza, and Christof Monz. 2017. Data augmentation for low-resource neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 567–573, Vancouver, Canada. Association for Computational Linguistics.

Steven Y. Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. 2021. A survey of data augmentation approaches for NLP. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 968–988, Online. Association for Computational Linguistics.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online. Association for Computational Linguistics.

Kunihiko Fukushima. 1975. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*, 20(3-4):121–136.

Artyom Gadetsky, Ilya Yakubovskiy, and Dmitry Vetrov. 2018. Conditional generators of words definitions. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 266–271, Melbourne, Australia. Association for Computational Linguistics.

William A. Gale and Kenneth W. Church. 1993. A program for aligning sentences in bilingual corpora. *Computational Linguistics*, 19(1):75–102.

Jonas Gehring, Michael Auli, David Grangier, and Yann Dauphin. 2017. A convolutional encoder model for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 123–135, Vancouver, Canada. Association for Computational Linguistics.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *arXiv preprint*.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified cross-modal pre-training for code representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7212–7225, Dublin, Ireland. Association for Computational Linguistics.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. GraphCodeBERT: Pre-training code representations with data flow. In *International Conference on Learning Representations*.

Mandy Guo, Qinlan Shen, Yinfei Yang, Heming Ge, Daniel Cer, Gustavo Hernandez Abrego, Keith Stevens, Noah Constant, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Effective parallel corpus mining using bilingual sentence embeddings. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 165–176, Brussels, Belgium. Association for Computational Linguistics.

Mandy Guo, Yinfei Yang, Keith Stevens, Daniel Cer, Heming Ge, Yun-hsuan Sung, Brian Strope, and Ray Kurzweil. 2019. Hierarchical document encoder for parallel corpus mining. In *Proceedings of the Fourth Conference on Machine Translation*

(Volume 1: Research Papers), pages 64–72, Florence, Italy. Association for Computational Linguistics.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Michael A. Hedderich, Lukas Lange, Heike Adel, Jannik Strötgen, and Dietrich Klakow. 2021. A survey on recent approaches for natural language processing in low-resource scenarios. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2545–2568, Online. Association for Computational Linguistics.

Felix Hill, Kyunghyun Cho, and Anna Korhonen. 2016a. Learning distributed representations of sentences from unlabelled data. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1367–1377, San Diego, California. Association for Computational Linguistics.

Felix Hill, Kyunghyun Cho, Anna Korhonen, and Yoshua Bengio. 2016b. Learning to understand phrases by embedding the dictionary. *Transactions of the Association for Computational Linguistics*, 4:17–30.

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.

David M. Howcroft and Dimitra Gkatzia. 2022. Most NLG is low-resource: here's what we can do about it. In *Proceedings of the 2nd Workshop on Natural Language Generation, Evaluation, and Metrics (GEM)*, pages 336–350, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.

Hamel Husain, Hongqi Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2020. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint*.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2018. Mapping language to code in programmatic context. In *Proceedings of the 2018 Con-*

ference on Empirical Methods in Natural Language Processing, pages 1643–1652, Brussels, Belgium. Association for Computational Linguistics.

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2017. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

Marcin Junczys-Dowmunt. 2018. Dual conditional cross-entropy filtering of noisy parallel corpora. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 888–895, Belgium, Brussels. Association for Computational Linguistics.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. 2018a. Marian: Fast neural machine translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia. Association for Computational Linguistics.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Shubha Guha, and Kenneth Heafield. 2018b. Approaching neural grammatical error correction as a low-resource machine translation task. 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 595–606, New Orleans, Louisiana. Association for Computational Linguistics.

Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. 2020. Learning and evaluating contextual embedding of source code. In *Proceedings of the 37th International Conference on Machine Learning*.

Dimitri Kartsaklis, Mohammad Taher Pilehvar, and Nigel Collier. 2018. Mapping text to knowledge graph entities using multi-sense LSTMs. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1959–1970, Brussels, Belgium. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, Conference Track*.

Tom Kocmi, Rachel Bawden, Ondřej Bojar, Anton Dvorkovich, Christian Federmann, Mark Fishel, Thamme Gowda, Yvette Graham, Roman Grundkiewicz, Barry Haddow, Rebecca Knowles, Philipp Koehn, Christof Monz, Makoto Morishita, Masaaki Nagata, Toshiaki Nakazawa, Michal Novák, Martin Popel, and Maja Popović. 2022. Findings of the 2022 conference on machine translation (WMT22). In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 1–45, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.

Philipp Koehn, Huda Khayrallah, Kenneth Heafield, and Mikel L. Forcada. 2018. Findings of the WMT 2018 shared task on parallel corpus filtering. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 726–739, Belgium, Brussels. Association for Computational Linguistics.

Philipp Koehn and Rebecca Knowles. 2017. Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 28–39, Vancouver. Association for Computational Linguistics.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Ashutosh Kumar, Satwik Bhattamishra, Manik Bhandari, and Partha Talukdar. 2019. Submodular optimization-based diverse paraphrasing and its effectiveness in data augmentation. 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 3609–3619, Minneapolis, Minnesota. Association for Computational Linguistics.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature*, 521(7553):436–444.

Chongman Leong, Derek F. Wong, and Lidia S. Chao. 2018. UM-pAligner: Neural network-based parallel sentence identification model. In *11th Workshop on Building and Using Comparable Corpora*, Paris, France. European Language Resources Association (ELRA).

Bohan Li, Yutai Hou, and Wanxiang Che. 2022. Data augmentation approaches in natural language processing: A survey. *AI Open*, 3:71–90.

Haonan Li, Fajri Koto, Minghao Wu, Alham Fikri Aji, and Timothy Baldwin. 2023a. Bactrian-x: A multilingual replicable instruction-following model with low-rank adaptation. *arXiv preprint*.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Luke Zettlemoyer, Omer Levy, Jason Weston, and Mike Lewis. 2023b. Self-alignment with instruction backtranslation. *arXiv preprint*.

Zhenhao Li and Lucia Specia. 2019. Improving neural machine translation robustness via data augmentation: Beyond back-translation. In *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*, pages 328–336, Hong Kong, China. Association for Computational Linguistics.

Yijiang Lian, Zhijie Chen, Jinlong Hu, Kefeng Zhang, Chunwei Yan, Muchenxuan Tong, Wenying Han, Hanju Guan, Ying Li, Ying Cao, Yang Yu, Zhigang Li, Xiaochun Liu, and Yue Wang. 2019. An end-to-end generative retrieval method for sponsored search engine–decoding efficiently into a closed target domain. *arXiv preprint*.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *arXiv preprint*.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou,

Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Timothee Mickus, Denis Paperno, and Matthieu Constant. 2019. Mark my word: A sequence-to-sequence approach to definition modeling. In *Proceedings of the First NLPL Workshop on Deep Learning for Natural Language Processing*, pages 1–11, Turku, Finland. Linköping University Electronic Press.

Timothee Mickus, Kees Van Deemter, Mathieu Constant, and Denis Paperno. 2022. Semeval-2022 task 1: CODWOE – comparing dictionaries and word embeddings. In *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*, pages 1–14, Seattle, United States. Association for Computational Linguistics.

Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, Workshop Track*.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. 2022. Cross-task generalization via natural language crowdsourcing instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3470–3487, Dublin, Ireland. Association for Computational Linguistics.

Yuya Morinaga and Kazunori Yamaguchi. 2018. Improvement of reverse dictionary by tuning word vectors and category inference. In *International Conference on Information and Software Technologies*, pages 533–545, Cham. Springer International Publishing.

Niklas Muennighoff, Thomas Wang, Lintang Sutawika, Adam Roberts, Stella Biderman, Teven Le Scao, M Saiful Bari, Sheng Shen, Zheng Xin Yong, Hailey Schoelkopf, Xiangru Tang, Dragomir Radev, Alham Fikri Aji, Khalid Almubarak, Samuel Albanie, Zaid Alyafeai, Albert Webson, Edward Raff, and Colin Raffel. 2023. Crosslingual generalization through multitask finetuning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume*

1: Long Papers), pages 15991–16111, Toronto, Canada. Association for Computational Linguistics.

Dragos Stefan Munteanu and Daniel Marcu. 2002. Processing comparable corpora with bilingual suffix trees. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 289–295. Association for Computational Linguistics.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Guúlçehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany. Association for Computational Linguistics.

Thanapon Noraset, Chen Liang, Larry Birnbaum, and Doug Downey. 2017. Definition modeling: Learning to define word embeddings in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3259–3266.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Mohammad Taher Pilehvar. 2019. On the importance of distinguishing word meaning representations: A case study on reverse dictionary mapping. 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 2151–2156, Minneapolis, Minnesota. 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, New Orleans, Louisiana. Association for Computational Linguistics.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the*

Association for Computational Linguistics: Volume 2, Short Papers, pages 157–163, Valencia, Spain. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Alessandro Raganato and Jörg Tiedemann. 2018. An analysis of encoder representations in transformer-based machine translation. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 287–297, Brussels, Belgium. Association for Computational Linguistics.

Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. 2020. COMET: A neural framework for MT evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702, Online. Association for Computational Linguistics.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, M. Zhou, Ambrosio Blanco, and Shuai Ma. 2020. CodeBLEU: a method for automatic evaluation of code synthesis. *arXiv preprint*.

Philip Resnik. 1999. Mining the web for bilingual text. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 527–534, College Park, Maryland, USA. Association for Computational Linguistics.

Philip Resnik and Noah A. Smith. 2003. The web as a parallel corpus. *Computational Linguistics*, 29(3):349–380.

Baptiste Roziere, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised translation of programming languages. In *Advances in Neural Information Processing Systems*, volume 33, pages 20601–20611. Curran Associates, Inc.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Lisbon, Portugal. Association for Computational Linguistics.

Gözde Gül Şahin. 2022. To augment or not to augment? a comparative study on text augmentation techniques for low-resource NLP. *Computational Linguistics*, 48(1):5–42.

Víctor M. Sánchez-Cartagena, Marta Bañón, Sergio Ortiz-Rojas, and Gema Ramírez. 2018. Prompsit's submission to WMT 2018 parallel corpus filtering shared task. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 955–962, Belgium, Brussels. Association for Computational Linguistics.

Víctor M. Sánchez-Cartagena, Miquel Esplà-Gomis, Juan Antonio Pérez-Ortiz, and Felipe Sánchez-Martínez. 2021. Rethinking data augmentation for low-resource neural machine translation: A multi-task learning approach. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8502–8516, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M Rush. 2022. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*.

Holger Schwenk. 2018. Filtering and mining parallel data in a joint multilingual space. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 228–234, Melbourne, Australia. Association for Computational Linguistics.

Holger Schwenk, Vishrav Chaudhary, Shuo Sun, Hongyu Gong, and Francisco Guzmán. 2021. WikiMatrix: Mining 135M parallel sentences in 1620 language pairs from Wikipedia. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1351–1361, Online. Association for Computational Linguistics.

Holger Schwenk and Matthijs Douze. 2017. Learning joint multilingual sentence representations with neural machine translation. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 157–167, Vancouver, Canada. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016a. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016b. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Rico Sennrich and Martin Volk. 2011. Iterative, MT-based sentence alignment of parallel texts. In *Proceedings of the 18th Nordic Conference of Computational Linguistics (NODALIDA 2011)*, pages 175–182, Riga, Latvia. Northern European Association for Language Technology (NEALT).

Lei Shi, Cheng Niu, Ming Zhou, and Jianfeng Gao. 2006. A DOM tree alignment model for mining parallel data from the web. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 489–496, Sydney, Australia. Association for Computational Linguistics.

Hiroki Shimanaka, Tomoyuki Kajiwara, and Mamoru Komachi. 2018. RUSE: Regressor using sentence embeddings for automatic machine translation evaluation. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 751–758, Belgium, Brussels. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826. IEEE Computer Society.

Aleš Tamchyna and Ondřej Bojar. 2015. What a transfer-based system brings to the combination with PBMT. In *Proceedings of the Fourth Workshop on Hybrid Approaches to Translation (HyTra)*, pages 11–20, Beijing. Association for Computational Linguistics.

Brian Thompson and Philipp Koehn. 2019. Vecalign: Improved sentence alignment in linear time and space. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1342–1348, Hong Kong, China. Association for Computational Linguistics.

Sindhu Tipirneni, Ming Zhu, and Chandan K. Reddy. 2022. Structcoder: Structure-aware transformer for code generation. *arXiv preprint*.

Ashok Urlana, Pinzhen Chen, Zheng Zhao, Shay Cohen, Manish Shrivastava, and Barry Haddow. 2023. PMIndiaSum: Multilingual and cross-lingual headline summarization for languages in India. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 11606–11628, Singapore. Association for Computational Linguistics.

Jakob Uszkoreit, Jay Ponte, Ashok Popat, and Moshe Dubiner. 2010. Large scale parallel document mining for machine translation. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 1101–1109, Beijing, China. Coling 2010 Organizing Committee.

Dániel Varga, Péter Halácsy, András Kornai, Viktor Nagy, László Németh, and Viktor Trón. 2005. Parallel corpora for medium density languages. *Proceedings of the RANLP 2005 Conference*, pages 590–596.

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 *Advances in Neural Information Processing Systems*, pages 6000–6010.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy. Association for Computational Linguistics.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508, Toronto, Canada. Association for Computational Linguistics.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2022. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.

Jason Wei, Chengyu Huang, Shiqi Xu, and Soroush Vosoughi. 2021. Text augmentation in a multi-task view. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2888–2894, Online. Association for Computational Linguistics.

Jason Wei and Kai Zou. 2019. EDA: Easy data augmentation techniques for boosting performance on text classification tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6382–6388, Hong Kong, China. Association for Computational Linguistics.

John Wieting, Kevin Gimpel, Graham Neubig, and Taylor Berg-Kirkpatrick. 2019. Simple and effective paraphrastic similarity from parallel translations. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4602–4608, Florence, Italy. Association for Computational Linguistics.

Ziang Xie, Sida I. Wang, Jiwei Li, Daniel Lévy, Aiming Nie, Dan Jurafsky, and Andrew Y. Ng. 2017. Data noising as smoothing in neural network language models. In *5th International Conference on Learning Representations, Conference Track Proceedings*.

Hang Yan, Xiaonan Li, Xipeng Qiu, and Bocao Deng. 2020. BERT for monolingual and cross-lingual reverse dictionary. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4329–4338, Online. Association for Computational Linguistics.

Ilsun You and Kangbin Yim. 2010. Malware obfuscation techniques: A brief survey. In *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, pages 297–300.

Shiwen Yu, Ting Wang, and Ji Wang. 2022. Data augmentation by program transformation. *Journal of Systems and Software*, 190:111304.

Jiajun Zhang and Chengqing Zong. 2016. Exploiting source-side monolingual data in neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545, Austin, Texas. Association for Computational Linguistics.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. BERTScore: Evaluating text generation with BERT. In *International Conference on Learning Representations*.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Xuan Zhang and Kevin Duh. 2021. Approaching sign language gloss translation as a low-resource machine translation task. In *Proceedings of the 1st International Workshop on Automatic Translation for Signed and Spoken Languages (AT4SSL)*, pages 60–70, Virtual. Association for Machine Translation in the Americas.

Wei Zhao, Maxime Peyrard, Fei Liu, Yang Gao, Christian M. Meyer, and Steffen Eger. 2019. MoverScore: Text generation evaluating with contextualized embeddings and earth mover distance. In *Proceedings of the 2019 Conference on Empirical Methods*

in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 563–578, Hong Kong, China. Association for Computational Linguistics.

Lei Zheng, Fanchao Qi, Zhiyuan Liu, Yasheng Wang, Qun Liu, and Maosong Sun. 2020. Multi-channel reverse dictionary model. In *AAAI Conference on Artificial Intelligence*, pages 312–319. AAAI Press.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, LILI YU, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. 2023. LIMA: Less is more for alignment. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Pierre Zweigenbaum, Serge Sharoff, and Reinhard Rapp. 2017. Overview of the second BUCC shared task: Spotting parallel sentences in comparable corpora. In *Proceedings of the 10th Workshop on Building and Using Comparable Corpora*, pages 60–67, Vancouver, Canada. Association for Computational Linguistics.

Pierre Zweigenbaum, Serge Sharoff, and Reinhard Rapp. 2018. Overview of the third BUCC shared task: Spotting parallel sentences in comparable corpora. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Paris, France. European Language Resources Association (ELRA).