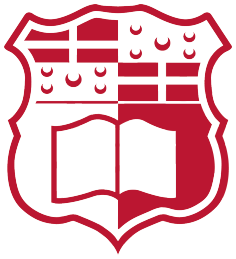

Error-Checking for Maltese

Deep Learning for a Low-Resource Scenario

Aaron Debattista

Supervised by Dr Claudia Borg

Co-supervised by Dr Ingrid Vella

Department of Artificial Intelligence

Faculty of ICT

University of Malta

October, 2022

A dissertation submitted in partial fulfilment of the requirements for the degree of M.Sc. in A.I.

L-Università ta' Malta

WWW.UM.EDU.MT

First edition, Monday 17th October, 2022

To my beautiful dog, Hazel

Who's a good girl?! YOU ARE!

Acknowledgements

I would like to express my gratitude to my supervisor and co-supervisor, Dr. Claudia Borg and Dr. Ingrid Vella, for their guidance, patience and dedication. This project would not have been possible without them.

I would also like to thank my partner and family for their support throughout this journey. I am a difficult man to live with, and I am sure that it must not have been easy for them!

Special mention goes out to my band mates: Bertu, Gabriel, Simon and Jesmar. Our music, like this dissertation, is a testament to the love that exists in our hearts for the Maltese language.

Abstract

In this study, we deep dive into the topic of resource-scarce Grammar Error Correction (GEC). Already a vast subject, GEC adds new layers of complexity when applied to low-resource environments. We explore the state-of-the-art and how it stemmed from rudimentary rule-based systems, transitioned into statistical models, and eventually evolved into neural-based techniques. The deep learning solutions applied in GEC are encapsulated by Neural Machine Translation.

We experimented with Encoder-Decoder models, focusing on the Sequence-to-Sequence and transformer architectural paradigms. We implemented a solution based on these paradigms and established three baseline models: a Seq2Seq model, a Vaswani-style transformer (Vaswani et al., 2017) and a Nematus-style transformer (Sennrich et al., 2017). We then applied several literature-backed adaptations to improve its performance in a low-resource environment. During experimentation, we observed progressive improvement upon applying tied embeddings and pretrained models for transfer learning.

We evaluated the model against the Maltese language to observe the solution's performance when applied to a real-world low-resource language. The final model achieved a $F_{0.5}$ score of 31.84%. The score is reminiscent of other GEC solutions, particularly those submitted for the BEA-2019 shared task. However, the implications of this result need to be analysed critically. The other submitted systems in BEA-2019 had been trained on different datasets and under different conditions. Therefore, the score of 31.84% is not a conclusive indication that our solution is better/worse than the alternatives that were submitted during the shared task.

Nevertheless, we believe that the findings of this study are essential because Maltese still lags in terms of Grammar Error Correction and new innovative ways of bypassing the data shortages are required to push Maltese GEC to the next level.

Contents

List of Figures

List of Tables

List of Abbreviations

Adagrad Adaptive Gradient Algorithm 60
BEA-2019 Building Educational Applications 2019 28
BERT Bidirectional Encoder Representations from Transformers 31
BLEU BiLingual Evaluation Understudy 20
BPE Byte-Pair Encoding 53
BPTT Back-Propagation Through Time 17
CLEC Chinese Learner Error Corpus 14
CNN Convolutional Neural Network 17
CoNLL Conference on Natural Language Learning 24
ESL English as a Second Language 11
EncDec Encoder-Decoder 3
ERRANT ERRor ANnotation Toolkit 30
GEC Grammar Error Correction v
GED Grammar Error Detection 11
GLEU Generalised Language Evaluation Understanding 24
GPU Graphical Processing Unit 34
GRU Gated Recurrent Unit 5
HOO Helping Our Own 28
ICC Intraclass Correlation Coefficient 83
LOCNESS The Louvain Corpus of Native English Essays 29
LSTM Long Short-Term Memory Cell 5
MLE Mean Likelihood Estimation 28
MLRS Maltese Language Resource Server 3

1

Introduction

In Natural Language Processing (NLP), GEC encapsulates all plausible methods for converting text from an ungrammatical form into a grammatical one. There is a labyrinth of different techniques and tools, and many have been continuously developing since the discipline's inception. Numerous avenues for implementation exist that include complex rule-based systems, statistical solutions, intelligent classifiers and deep-learning enhanced models. The development of these systems is further complicated when considering the numerous distinctions between different languages.

In the present day, there are over 7000 (Kibrik, 2008) languages. Each natural language contains thousands of years of historical adaptation and cultural influences. Geographical differences of a few kilometres have spawned intricate dialects in the same tongues. Written language follows suit, and experiences change as well. The diversity skyrockets when considering how different cultures adapted similar sets of characters to represent different phonetics and pronunciations. These differences create the unique challenges to implementing GEC, as a one-fits-all solution is not realistically feasible.

Modern GEC implementations are more data-dependent than ever. Adding to the long list of differences between languages, we must also consider that not every language has a similar electronic presence. Languages like English and French have benefited from globalisation, and it is much easier to gather the required data resources to train complete GEC systems. The same does not apply to several other languages that lack these resources entirely. Thus, low-resource GEC can be considered its own unique challenge and consists of using techniques that were traditionally always empowered with data in situations where these resources are nonexistent. Novel approaches have been developed that push every last bit out of GEC architectures to create complete solutions for low-resource settings.

1.1 | The Problem Area

Authoring an expert system comprised of programmed rules is possible. However, achieving this is difficult, especially considering that semantics influence every natural language. The spelling of any word within a sentence is ultimately contingent on the intended meaning behind the entire sentence. It is practically impossible to write a rules engine that is large, diverse and complex enough to model every possible utterance. At the very least, languages evolve, and the spelling of words could also change with demographic trends. Often, completely new words are inducted into the language, whether with the exact spelling or slightly altered spelling that brings it closer to its new lingual home.

On the flip side, there are options for addressing the problem using data-driven approaches. Researchers can train statistical and neural systems on real-world data, with the resulting solution gaining an edge in contextual awareness. Practices such as these would become the driving force behind the advancement of the state-of-the-art. Alas, they all shared a common bottleneck. For any data-driven solution to be effective, it must have adequate data to work with and train. The abundance of training material is not an insurmountable challenge for languages with a long-standing virtual presence. Systems written for English and French have vast training datasets from many different sources. The reality is different for the Maltese language.

Maltese is a language originating from Siculo-Arabic. It is a language based on a Semitic structure with a heavy presence of Romance lingual elements and some Anglo-Saxon influences. The word's root governs lexemes and inflexions because in Maltese, the root's consonants are a point of emphasis. Inflexions are modulations applied to words that express grammatical functions such as subject gender and tense. In most cases, tense and plurality variations do not affect these consonants' positions. Patterns on the base root form can determine morphology. Maltese is predominantly spoken phonetically. The sounds of the vowels and consonants generally do not change, and there is a very close relationship between the written and spoken language. Maltese also incorporates borrowed words, which later entered the language. The Maltese language inducted borrowed words by changing the spelling to produce the same pronunciation (Rosner et al., 2012). Approximately 400,000 people speak it. Therefore, compared to other languages, it not only has a relatively tiny speaking/writing population but is also inadequately resourced. The volumes of available Maltese corpora number only in the few tens of thousands of sentences. In contrast, millions of training instances exist for other more significant languages. Lastly, the computing power for large-scale model training can be rather demanding.

1.2 | Motivation

The path toward enhancing the language resources for Maltese has been long and time-consuming. Today, several research initiatives have left valid contributions to the language. Efforts toward delivering a spell checker have been happening as early as 2000. People can still discover some of the results of these efforts online (Casha, 2004). For example, the online spell checker by Casha uses a generated list of words and applies the same algorithm as that of *Unix Aspell*. However, this system was mainly a dictionary-based correction system and was not intended or even able to correct grammatical errors. Later on, Rosner et al. (2012) expanded on the existing resources by increasing the sizes of the available corpora on the Maltese Language Resource Server (MLRS)[1]. MLRS is the only online environment with accessible Maltese corpora and language resources on request. There have also been studies conducted to create a Part-of-Speech (PoS) tagger and a dependency parser for Maltese (Zammit, 2018), and a tagged corpus exists on the MLRS server (Gatt and Čéplö, 2013).

Besides the rule-based error correction system available online[2], there are no other alternatives for Maltese error correction. For a very long time, the state-of-the-art in Grammar Error Correction required large datasets that probabilistic phrase-based prediction systems could use to train. In contrast, deep learning methods have been steadily advancing and now feature new state-of-the-art options that do not require as much data as statistical systems. Such examples include Encoder-Decoder (EncDec) models such as the Sequence-to-Sequence (Seq2Seq) model (Bahdanau et al., 2014; Sutskever et al., 2014) and the transformer model (Sennrich et al., 2017; Vaswani et al., 2017). The efficacy of these architectures has been proven in low-resource scenarios and for obscure languages. Furthermore, several different data-level preprocessing techniques could realistically improve the output of a trained GEC model in a low-resource setting. These include parameter dropout, source word corruption, error synthesis, tied embeddings and transfer learning from pretrained models.

In this study, we thoroughly analysed the available options and investigated varying deep-learning architectures that could create a GEC tool for the Maltese language. We believe this attempt is the first to make an unsupervised deep-learning GEC solution for Maltese. Furthermore, this is the first time that techniques such as error synthesis are applied and evaluated in the setting of Maltese error correction. As with many of the past NLP research efforts in Maltese, this study exists with the primary purpose of appending to the slowly growing cadre of language resources in this domain. If it

[1] https://mlrs.research.um.edu.mt/
[2] https://spelling.mt/

succeeds, it will create a new baseline system that serves as the springboard upon which future researchers can initiate new studies centred around deep-learning approaches to GEC. It will inspire deeper evaluations and further improvements. In the event of failure, it can at the very least paint a clearer picture of the challenges faced by the Maltese language in such implementations and produce analytical data that could guide future studies. We embarked upon this research with confidence in its novelty and its contributions.

1.3 | Aims & Objectives

In this research, we aim to deliver an error correction solution which uses techniques applicable to low-resource settings. To achieve this goal, we sought to use Maltese (a low-resource language) as our basis for experimentation and evaluation.

We define three primary objectives in support of our aim.

1.3.1 | First Objective

Evaluate existing state-of-the-art deep-learning methods for grammatical error correction and implement a baseline model using a neural-based approach.

We achieve this objective by thoroughly examining the methods which yielded the most substantial results for GEC. We seek to identify the highest quality solutions in terms of standard scoring metrics (such as BLEU and F-Score). We will then replicate the best solutions that we discover in the literature. After replication, we seek to reproduce the original paper's results and verify the authenticity of any claims. Furthermore, after replicating, we would arrive at a starting point for this study. We take the knowledge from the replication and deliver our baseline models, which will be more closely related to our subject domain.

1.3.2 | Second Objective

Augment the Grammar Error Correction model using supported techniques that are meant for resource-limited settings.

We achieve this objective by employing research-backed augmentations to the model. Many techniques allow for more substantial results in resource-scarce environments. We aim to adapt the resulting baseline models from the first objective. We will then determine how much the quality of the baseline models would improve, if at all.

1.3.3 | Third Objective

Apply the model in the context of the Maltese language and evaluate it using a similar methodology to other related neural-based systems in environments with resource scarcity.

After completing the first two objectives, we aim to use the new system to correct Maltese text. We will achieve this by evaluating the system on Maltese text corpora. We will evaluate the resulting model qualitatively by applying it to real-world text examples containing errors. We will conclude by analysing the impact of the adaptations introduced in the second objective.

1.4 | Approach

To achieve the defined scope, we focus our research on the GEC candidates most likely to be successful. The literature championed the adoption of EncDec models (Cho et al., 2014; Kalchbrenner and Blunsom, 2013). EncDec models occupy a prominent position in the state-of-the-art for making generalisations on unsupervised training data. They are built using deep learning architectures. The strongest EncDec models are typically consistently implemented using a Recurrent Neural Network (RNN) (Graves, 2013), but either Long Short-Term Memory Cell (LSTM) cells (Cho et al., 2014) or Gated Recurrent Unit (GRU) cells (Etoori et al., 2018; Junczys-Dowmunt et al., 2018b) could comprise the arrays of cells that constitute these networks.

As far as neural-based GEC is concerned, the process is no different to regular language translation. In both cases, one converts a source language to a target language. The only difference is that in GEC, the source language is not a different language but a grammatically incorrect variant of the target language. In the EncDec architecture, translation occurs by ferrying input sequences to the encoder, encoding them into numerical vectors, ferrying their output to the decoder and then decoding that information back into human-readable format. The behaviour of the process earned the model the additional moniker of *Sequence-to-Sequence model*. Another feature that drastically reduced hardware requirements, referred to as an *attention mechanism* (Vaswani et al., 2017), enhanced the design of Seq2Seq models. Seq2Seq models that included this mechanism were referred to as *transformers*.

We include some adaptations to the models to improve their performance further. These are all inspired by the work of Junczys-Dowmunt et al. (2018b) and comprise the following:

- Dropout: A technique that removes neurons from the layers. Dropout helps the network not become overly dependent on them.
- Source Word Corruption: A technique whereby words are removed from the source files to allow the GEC system to train without becoming dependent on the presence of all the source tokens in the source sequences.
- Error Synthesis: Synthesising and manually introducing errors into valid texts. Synthesis accounts for the lack of sentence pairs in the data and artificially increases the volumes that the model can use for training.
- Tied Embeddings: Improves the performance of the translation model by sharing the weights of the embedding layer and the layers with the activation functions (often referred to as Softmax layers). This method improves the quality of the model and reduces the number of input parameters required by the model. This method also reduces hardware requirements and enhances efficiency.
- Pretrained Model Transfer Learning: Using a pretrained model to initialise the neural network's weights.

We train the system on sanitised files that contain Maltese text. We also evaluate the system against never-seen-before real-world documents used in a Maltese proofreading course.

1.5 | Document Structure

The dissertation is structured as follows.

1.5.1 | Introduction

This chapter briefly introduces the problem area and provides a synopsis of where we find ourselves positioned regarding delivering a GEC tool for the Maltese language. We describe the aim and objectives in some detail.

1.5.2 | Background & Literature Review

This chapter contains valuable background information and in-depth exploration of the literature related to our topic. It explores several different systems and paradigms for implementing Grammar Error Correction. Later in the chapter, we focus on Neural Machine Translation as the most potent option for implementing GEC within a low-resource setting. We were further emboldened to champion these neural techniques thanks to Encoder-Decoder models and their firm unanimous position in the state-of-the-art of error correction. The latest technological advancements, such as the attention mechanism, were a critical factor that made EncDec architectures a viable option even if hardware resources were scarce. Lastly, we identify the BEA-2019 shared task, an event that generated a lot of research effort in low-resource GEC. The submissions for this shared task were on the cutting edge of the state-of-the-art and served as the best seeds of inspiration and benchmarks for evaluation.

1.5.3 | Implementation

This chapter describes our chosen implementation, which was heavily supported by the literature. We brought together a collection of relevant datasets organised in sentence pairs, where each ungrammatical sentence in the former file had its aligned grammatical counterpart in the latter file. We then implement a series of baseline Encoder-Decoder models to compare the quality of Seq2Seq models against the quality of transformers with the attention mechanism. We focus on the best-performing model and added several adaptations to improve its performance in low-resource settings. We implement all models with the renowned Marian NMT toolkit (Junczys-Dowmunt et al., 2018a). Supporting the training process was a set of auxiliary systems, including a fully custom-built error synthesiser, a tokeniser and a short script that could translate BERT models into a format that was readable by Marian NMT.

1.5.4 | Results

In this chapter, we conduct experiments to gain valuable insights into the quality of the competing NMT architectures. In particular, we focus on a traditional Seq2Seq model, a transformer model based on Vaswani et al. (2017) and another transformer model implementation based on Nematus (Sennrich et al., 2017), named AMUN. We champion one of them and then apply a series of different adaptations that the literature recommends to improve the efficacy of the final model. Adaptations include source word corruption, domain and error adaptation, tied embeddings and pretraining. We train

our models from two Maltese language models built on BERT (Micallef et al., 2022), one monolingual while the other multilingual. We compare and contrast the results of our findings.

1.5.5 | Evaluation

In this chapter, we take our best-performing models from the previous chapter and evaluate them against new, never-before-seen data. We wanted to assess our results using methods that participants used for the BEA-2019 shared task. We base many of our decisions on approaches that had been successful in that shared task. We use the ERRANT scorer to compute the models' precision, recall and $F^{0.5}$ scores. We then discuss our findings, the issues faced by the solution and how it compares with systems that had participated in the BEA-2019 shared task.

1.5.6 | Conclusions

In this chapter, we revisit our journey during this study's compilation. We determine whether or not we had achieved the goals we set out to accomplish and comment on the results we obtained. We detail the limitations of our endeavour and recommend ways in which future researchers could continue building on our work.

2

Background & Literature Review

This chapter presents the various methodologies that past researchers had outlined. The literature revealed that while adopting specific methods followed a vaguely chronological order, the reality depicts that different techniques competed against each other, often in parallel. The growth, maturity and decline of varying GEC paradigms coexisted. The novelty of these systems often lies in combining the most potent elements from varying methods, thus creating complex hybrid systems. Therefore, we structured the review around the main categories and strategies of GEC. We arranged the content to reflect the order in which these systems rose to prominence as the field evolved.

As a complete process, GEC involves two logical phases. The first is *grammar detection* and the second is *grammar correction*. These phases and their distinction at a technical level became less pronounced as the field advanced. Several different kinds of errors could appear in natural language text, and the efficacy of repairing such errors varies depending on the implementation choice.

The chapter is organised as follows. Section 2.1 is a brief background about error types. Section 2.2 describes the first correction systems based on dictionaries and rule engines. Section 2.3 details the rise in prominence of systems that relied on statistical engines. Section 2.4 describes the brief stint of error classifiers which were eventually surpassed by neural methods discussed in Section 2.5. We describe some hybrid systems in Section 2.6 and elaborate how Neural Machine Translations (NMTs) were applied to GEC in Section 2.7. The challenges related to resource scarcity are described in Section 2.8 and Section 2.9 is dedicated to discussing the BEA-2019 Shared Task which was set up to generate research interest in Grammar Error Correction, with keen attention also given to finding a solution to the resource scarcity problem. Section 2.10 contains a short summary of the entire chapter.

2.1 | Error Types

The earliest solutions did little more than check references from hash tables. When dealing with simple error types such as *non-word errors*, dictionary-based and rule-based systems would perform the job sufficiently. Since non-word errors do not exist in the source grammar, it would be a matter of recognising clear spelling violations (Ahmed et al., 2009). However, systems akin to these were simple spell checkers rather than pure GEC implementations. True GEC systems would not come to fruition until researchers started addressing the issue of *context*. Context refers to the intended meaning behind a word or sentence (Kukich, 1992). It is possible to write sentences with correct spelling but incorrect grammar. For example, 'I am note a morning person' is syntactically correct but has incorrect grammar. Errors such as these are categorised as *real-world errors* (Ahmed et al., 2009). Sometimes, a real-world error might not impact the grammar but would morph the sentence's intended meaning. For example, 'I am not a mourning person' is both syntactically and grammatically correct, but the semantic meaning of the sentence does not match the speaker's intention. These are more challenging to discover (Samanta and Chaudhuri, 2013).

Non-word errors and real-word errors are further classified into *typographic* and *cognitive* errors. Typographic errors result from erroneous keyboard input, and we can describe them in terms of keyboard key proximity (Kaur and Singh, 2015). For example, one could misspell 'night' as 'nighr' (a non-word) or 'might' (a real word). We can measure these errors in terms of insertions, substitutions, deletions and transpositions (Gill and Lehal, 2007). Cognitive errors occur due to misunderstanding how a word should be spelt. For example, one could write the word 'kite' as 'kight' (a non-word cognitive error) due to the spelling of similarly pronounced words such as 'knight' and 'fight'. This example is referred to as a *phonetic* error since the letter sequence has the same sound as the intended word. One could also err by writing one phonetically similar word instead of another, such as 'night' instead of 'knight' (a real-world cognitive error). There is no syntactic error in this case, but the input should still be considered incorrect. Such instances are referred to as *homonym* errors (Ahmad and Kondrak, 2005).

2.2 | Dictionary & Rule-Based Techniques

One of the earliest rule-based systems was *Spell* (Kukich, 1992). Spell was an open-source grammatical error detection program designed for Unix. It worked by executing a lookup operation on each string token and referencing it against a massive word dic-

tionary. One could augment the dictionary to become a better-tailored fit for the knowledge domain described in the text. The program could also account for inflexions that change the spelling of a word. Unix Spell was purely a Grammar Error Detection (GED) system. It provided users with a list of suggestions and left it to them to decide which alternative was best to replace the erroneous string. Spell did not provide any correction capabilities. The authors used additional Unix programs such as *Grope* to address this shortcoming. Grope assists user decision-making by ordering the suggestions according to a scoring metric. Eventually, more advanced open-source spell checking tools like *Ispell* and *Aspell* would replace Unix Spell, boasting larger dictionaries and grammar correction capabilities (Naber and Witt, 2003).

A system presented by Park et al. (1997) applied a grammar checker for *English as a Second Language (ESL)*. In this example, there was a greater emphasis on using the context of the sentence to identify specific kinds of errors. It could pinpoint capitalisation errors, missing prepositions, incorrect articles, wrong verb forms and incongruences between tenses and adverbs. Park et al.'s solution relied on hand-written rules, and a large *PoS* tagged lexicon supplemented by a morphology table. The study did not detail any evaluation metrics, though the system's biggest weakness was its inability to detect errors that the authors did not explicitly define in the rule base. The resulting system was better for dealing with errors commonly observable in ESL and did not offer any correction mechanisms.

Later, Naber and Witt (2003) developed a solution using similar foundations. It required the creation and maintenance of a rule base written in XML format. One could switch the rule base according to the domain or changes in the language. It worked by making use of PoS tags just like in the previous solution. The paper cited 93% recall and 100% precision though these results were of the PoS tagger and not of the GEC solution. The authors could not conduct proper precision and recall testing due to data shortages and limitations in the corpora.

Around the same time as Park et al. (1997), Tschichold et al. (1997) proposed a different approach for an ESL grammar checker meant for native French speakers. This solution relied on *finite state automata*. Rather than having a preset rule base, the authors modelled the language's grammar as a state machine that could recognise when text strings ended up in an erroneous state. It performed poorly, having detected approximately 14% for most error types. It was inferior to other systems established during that period, such as WinProof. However, they still managed to reduce the rate of error overflagging when compared to their peers. Like all of their predecessors mentioned thus far, the system was only capable of highlighting errors. The correction phase did not go beyond citing different recommendations for resolving the issues within the sentence.

All rule-based systems had the advantage of being relatively simple to implement and requiring no training to be functional. However, their greatest downfall was being ill-equipped to deal with varying grammatical contexts. Even systems like that of Park et al. (1997), which the authors wrote with special measures to account for contextual ambiguity, could never realistically account for all possible grammatical scenarios, especially when the sentence length and complexity were high.

At this point, the state-of-the-art started gravitating more towards data-driven alternatives. Thus, the field of machine translation was coming to force. Machine translation uses computer-based technologies to translate or map text from one language to another. We refer to the two languages as the source and target languages. GEC can be an example of such a translation task. Rather than translating between two different languages, the source and target languages in a GEC system are the grammatically incorrect and grammatically correct versions of the exact text. At a machine translation level, language translation and grammar correction are functionally the same (though the challenges are still different). Among the first machine translation models to come to fruition were those based on statistical data (Brown et al., 1993).

2.3 | Statistical Machine Translation

Statistical Machine Translation (SMT) is a machine translation paradigm that uses statistical methods. Unlike rule-based systems, SMT systems have no hard-coded error-handling mechanisms. Instead, they rely on specially built Bayesian probabilistic models to correct grammatical errors. These models are often based on extensive collections of training data to be effective and are trained with rows of incorrect-correct sentence pairs. Upon detecting an error, the statistical model computes the most probable corrections. Provided that the necessary language corpora are used during training, SMT solutions can be both monolingual or multilingual.

In 1990, Brown et al. (1990) published a paper detailing a statistical approach toward machine translation. In this study, Brown et al. applied a statistical model for translations between English and French. Brown et al. distinguished between the *language model* and the *translation model*. The language modelling phase of a machine translation system involves the determination of the probability of words occurring in a sequence within a language. The language model aims to represent the target language and its grammatical correctness. For example, the phrase 'I go' should have a higher probability of occurring than 'Go I'.

On the other hand, translation modelling describes the modelling of the probability

that a sequence of tokens is an accurate translation of another. In the case of the translation model, the goal is that the most faithful translation between the source and target languages would have the highest probability. Theoretically, the translation of a phrase is explained using Bayes' Theorem. The mathematical calculation for this is shown in Equation 2.1, where S is the source sentence, T is the target sentence, and $Pr(S|T)$ represents the probability of a source sequence given a translated sequence.

$$Pr(S|T) = \frac{Pr(S)Pr(T|S)}{Pr(T)} \tag{2.1}$$

Given our observation T, i.e. a sentence in the target language, we seek to discover which sentence S (the hypothesis) produced that translation. We eliminate the denominator $Pr(T)$, which is independent of S and aim to maximise the product of $Pr(S)$ and $Pr(T|S)$. We therefore arrive to the fundamental equation of Statistical Machine Translation as shown in Equation 2.2 (Brown et al., 1993). The intuition involved in this determination can be seen as a channel whereby the passing of noisy words will yield the most probable correct translation. Therefore, this model is often described as a *noisy channel*.

$$\hat{S} = argmax\, Pr(S)Pr(T|S) \tag{2.2}$$

The authors also introduce the idea of *alignment*. Alignment describes the relationship between the words of the source and target languages. For example, the English phrase, 'the cat' would align with the French term, 'le chat'. The previous example is the simplest since, in more complex translations, aligned words would not necessarily line up with each other perfectly. For example, the English and French phrases 'is not' and 'n'est pas' would have an unconventional alignment whereby the token 'not' would be aligned to the tokens 'n' and 'pas', which surround the token 'est' (that aligns with 'is'). The algorithm worked by estimating the probability of translations and then using them to provide the alignments for the translation model.

The concept of the *Noisy Channel Model* was not only applicable in cross-lingual translation but also Grammar Error Correction. Like previously discussed rule-based approaches, researchers developed several SMT solutions to tackle the challenges faced in ESL applications. Brockett et al. (2006) developed an SMT system for correcting English text written by native Chinese speaking students. Due to the considerable variance in error types, Brockett et al. limited the scope of the system to only detecting *mass noun errors* (uncountable nouns such as 'information' or 'advice'). Like the study of Brown et al. (1993), this solution also relied on the noisy channel framework (Brown et al.,

1993; Kukich, 1992). The best target correction was based on noisy input obtained using scrambled letters and other incorrect spellings. The model then calculated the most likely correct candidate based on the probabilities of each mistake in the source text. In the case of the approach taken by Brockett et al. (2006), the authors sourced the training corpus from Reuters. However, the corpus was comprised completely of grammatically correct sentences. The noise had to be introduced into the dataset manually to fine-tune the system. Therefore, error patterns were sourced from the Chinese Learner Error Corpus (CLEC) and synthetically introduced into the Reuters corpus. The system could correct 61% of errors when tested against grammatically incorrect queries sourced from the web. Error synthesis as a means of training would become much more popularised in future GEC solutions, especially with the advent of neural-based techniques.

SMTs like the ones of Brown et al.'s and Brockett et al.'s were among the first solutions that could reliably recommend corrections to users rather than identify the error. Errors could be measured using various algorithms. Most commonly in SMTs and the Noisy Channel Model paradigm, differences between input words and their corrected counterparts were aggregated using string distance algorithms such as that of Damerau and Levenshtein (Felice et al., 2016), which tallied omissions, insertions, substitutions and transpositions.

String distance was used in the studies of Cucerzan and Brill (2004), and Ahmad and Kondrak (2005), both of which focused on training error models from query log statistics. Cucerzan and Brill discovered that 10-15% of search queries contained spelling errors which strengthened their potential application as a noisy training corpus. In Cucerzan and Brill's study, the authors tokenised the strings and compared each token against several alternative spellings computed according to different string distance thresholds. The thresholds were extracted from a set of unigrams and bigrams, and the system discovered the best possible alternatives using a search function. The system achieved a precision score of 88% and a recall score of 85%.

Ahmad and Kondrak's approach followed the same reasoning as Cucerzan's but also included a soft-clustering algorithm for calculating probabilities. The model was tested against established correction programs like *Ispell* and *Aspell*. Ahmad and Kondrak's *EMBED* system achieved 79% accuracy, surpassing Ispell but falling short of Aspell. This accuracy score showed when the solution found proper corrections in the suggested list of candidates. Realistically, the top-most recommended candidate was accurate only 41% of the time.

A later system by Samanta and Chaudhuri (2013) focused on correcting real-world errors. The authors built confusion sets for words containing all possible alterations the system could form following edit operations calculated using Levenshtein string dis-

tance. The surrounding bi-grams and tri-grams were also factored into calculating the probability of the occurring error. The system then passed this information to a weighting function to produce the final correction. The system achieved a precision score of 71% and a recall score of 81%. English was the primary test subject for GEC system, but the solution could also evaluate other languages since it used token neighbourhoods (bi-grams/tri-grams surrounding the token) to generate the probabilities.

SMTs were prevalent in other languages aside from English. Gill and Lehal (2007), Kashefi et al. (2013), and Kaur and Singh (2015) proposed systems for the Punjabi, Persian and Hindi languages, respectively. These systems relied on string distance algorithms to generate candidate suggestions, similarly to Samanta and Chaudhuri. The Punjabi and Hindi spell-checkers were limited since they could only detect typographic errors. The Persian spell-checker was more robust than the other two because it could also correct phonetic and keyboard key proximity errors. Kashefi et al. compared various string distance measures but proposed their unique customised alternative suited for the Persian language.

The biggest issue of statistical systems was their over-reliance on large datasets. For languages such as English, this was not a problem. Due to it being one of the most globally spoken languages, large corpora were highly available. However, this was not the case with other lesser-known languages. Researchers had to look for other alternatives. For some time, solutions for languages within low-resource settings had to resort to older and less powerful methods. For example, Naseem and Hussain (2007) implemented a rule-based approach for Urdu and Wasala et al. (2010) designed a GEC system for Sinhala, which used an algorithm that synthesised word permutations based on phonetic similarity as opposed to sourcing them from a corpus. The Urdu system was poorly evaluated and published no results, and the Sinhala system under-performed compared to an existing Sinhala GEC (based on a traditional SMT setup) by 34%.

2.4 | Error Classifiers

As the use of SMTs continued to mature and saturate, new innovative forms of error correction were starting to take shape. Researchers began to dabble in using classifiers simultaneously when SMTs dominated the state-of-the-art. The use of classifiers would signal a shift in the status quo away from probabilistic methods and closer to machine learning alternatives. Researchers experimented with nascent classifier-based techniques before eventually gravitating towards deep network translators. The stint of classifiers in Grammar Error Correction was relatively short-lived, but it yielded several

models designed to detect specific types of errors.

In the beginning, classifiers started to be used for resolving simple errors. The earliest example of such classifiers was that of Knight and Chander (1994), a decision-tree system that could only determine the correct choice between 'a' and 'an' by using *noun phrases*. A noun phrase constitutes any group of words that could replace a pronoun, which is also a noun phrase (for example, 'he', 'that man', and 'John' are all noun phrases). The system was trained using 400,000 noun phrases from the *Wall Street Journal* and achieved 89% accuracy. The use of PoS tags was not an isolated case, such as in Minnen et al. (2000), where a system was designed to correct articles. In this study, the authors trained a memory-based learning algorithm on the *Penn Treebank* data collection. The model worked by predicting the most likely article based on its trained instances. The determiner was identified using the PoS tags, including the noun phrase and other related tags. In its limited context, the model achieved an accuracy score of 83%, but this required 300,000 noun phrase observations for training.

In 2008, Tetreault and Chodorow described a methodology for detecting prepositional errors in ESL texts. The detection algorithm was implemented using a *Maximum Entropy Classifier* based on earlier work by Chodorow et al. (2007). A classifier trained on 7 million different preposition contexts. The resulting model was equipped to fix incorrect prepositions and identify prepositional errors, such as when the writer would introduce a preposition unnecessarily or write it twice by mistake. The original system (Chodorow et al., 2007) had achieved 79% precision and 11% recall when the authors evaluated it against ESL text that was unused during training. Additional to the baseline system, Tetreault and Chodorow applied combinational models to improve the results. These were simple structures built with PoS annotations used as training features in the model. These adaptations only improved the precision and recall by 2% and 3%, respectively.

In 2011, Rozovskaya and Roth (2011) evaluated several different classification systems. Like the previously discussed studies, the subject area was also ESL. The study determined the superiority of the *averaged perceptron*, a primitive version of the modern-day neural network. Additionally, the study forwarded its contribution by training a classifier on the correct prepositions and the incorrect prepositions found in the text. This technique is a vague homage to future studies that would train models using sentence pairs in neural solutions. The resulting model became better adapted for fixing incorrect prepositions due to introducing them into the training process. Using this adaptation, the authors managed nearly double the accuracy of a typical classifier (Rozovskaya and Roth, 2011).

When compared against SMTs, classifiers were reasonably accurate and precise.

However, they suffered from issues which were their inclination toward low recall rates. The range of errors they could frequently correct was limited, and they focused only on a minimal subset of error types. This limited scope was a considerable handicap compared to the later SMT systems such as the one of Felice et al. (2014) which could correct over 25 different error types, albeit to varying degrees of precision and recall. Classifiers also required vast amounts of data, and since most classifier-based solutions relied on supervised techniques, the datasets had to be annotated. Unfortunately, annotated data was a significant source of contention for low-resource environments. Annotated sets were not as abundant, making classifier-based solutions unsuitable for such scenarios.

Researchers started redirecting their efforts towards translation techniques that could yield accurate models without needing large amounts of labelled data. The studies in classifiers supported the effectiveness of deep learning methods due to their ability to better capture the context of sequences. Hence, the state-of-the-art started moving closer to deep learning methods. This move culminated in the conception of a novel form of machine translation called Neural Machine Translation (NMT).

2.5 | Neural Machine Translation

One of the very first instances of NMT-based systems was that of Kalchbrenner and Blunsom (2013) which used a recurrent network to capture continuous representations of text. Their solution was dubbed the Recurrent Continuous Translation Model (RCTM) and divided the problem into two training phases. The first phase was the creation of a *language model*. The language model's role was to create vector representations for the language sequences. The language model's sentence conditioning and subsequent creation were done using Convolutional Neural Networks (CNNs). The second phase was the translation phase, which entailed converting said representations into words belonging to the target language. The translation was achieved using a Recurrent Neural Network (RNN).

The Recurrent Neural Network is a sub-type of the artificial neural network organised in repeating cell modules. Referring to Figure 2.1, its distinctive design includes the weights of previous outputs with the rest of the inputs during training. The outcomes work as additional hidden layers within the neural network, and information stored in current epochs is continually influenced by the data saved during previous epochs. This characteristic distinguishes RNNs from traditional feed-forward neural networks, which enforce independence between the inputs and outputs of the model (Singh and Singh, 2020). Instead of Back-Propagation, RNNs use a variant algorithm named Back-

Propagation Through Time (BPTT). BPTT is functionally similar to Back-Propagation, but it differs in that errors are aggregated at each time step to account for shared parameters.

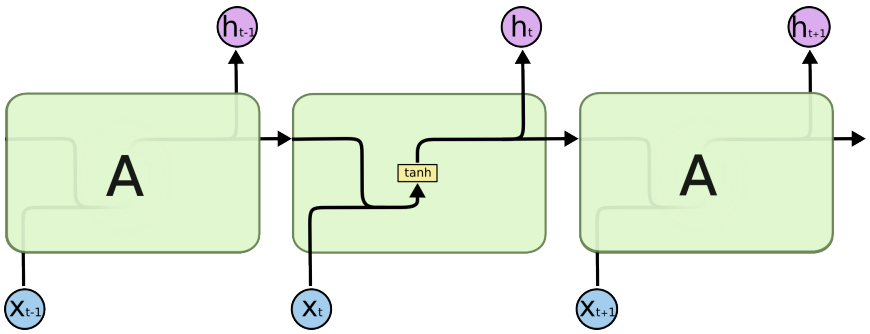

Figure 2.1: RNN
(Olah, 2015)

Neurons within an RNN are designed using normal activation functions such as SoftMax, RELU, Sigmoid and Tanh. Unlike traditional neural networks, recurrent networks are not constrained to map one input to one output and appear in one-to-many, many-to-one and many-to-many varieties. However, they still suffer from the vanishing gradient problem (Hochreiter and Schmidhuber, 1997). The vanishing gradient problem is a common issue in neural networks whereby the gradient becomes closer to zero with the addition of more hidden layers, and this behaviour ultimately delivers models which are difficult to train.

Kalchbrenner and Blunsom (2013) were the first to encode phrase representations into numeric information vectors and then map the vector back to a sentence during translation. Kalchbrenner and Blunsom's solution can be thought of as a prototype Encoder-Decoder model, though the actual term *EncDec* would be coined by a later study (Cho et al., 2014). Future models would be built on similar principles, though they would rely on network classes other than CNNs for sentence conditioning. The primary issue with convolutional networks was that they would lose too much information about how the output layer should order the tokens of the output sequence.

Encoder-Decoder models are comprised of 2 separate networks. The first network serves as the encoder, and it encodes a sequence of symbols by processing the tokenised series and reproducing a condensed information vector. Words can be represented mathematically (and programmatically) using vectors, and by extension, a deep learning network can thus represent a sequence of words with a series of vectors. The output of the encoder is then passed as input to the second network, i.e. the decoder. The latter network's job is to translate the vector representation back into symbols by

generating parts of the output sequence iteratively. In an Encoder-Decoder setup, the sentence meaning is encoded within these vector representations and conceptually exists between the encoder and the decoder. The encoder and the decoder are trained jointly to translate the same sentence meaning, thus maximising the chance of successful translation. Encoder-Decoder models are ideal for training on sentence pairs that have dissimilar grammar and token positioning (Cho et al., 2014).

In the study by Cho et al. (2014), the authors implemented a novel EncDec architecture using RNNs. The solution was trained from the Workshop on Statistical Machine Translation 2014 (WMT-2014) dataset and evaluated on an English-to-French translation task that was implemented using an existing SMT translation model. A novel aspect of this model was the use of Long Short-Term Memory Cells (LSTMs), which diminished previous issues encountered by setups that used CNN layers.

LSTMs were conceived by Hochreiter and Schmidhuber (1997) to counteract the negative setbacks due to information loss. When an input sequence becomes too long, the network will start to forget the context and meaning of that input because a phrase's contextual information might be supported by tokens that are quite further away. These instances are referred to as long-term dependencies and lend their name to the LSTM architecture. While standard RNNs are capable of processing past data, this data would still have to be relatively recent. Architecturally, RNNs are not robust enough to recall information from many sentences ago. However, LSTMs employ memory cells in the hidden layers. These cells have three gates - an input, an output and a forget gate. These three gates are denoted in Figure 2.2 by i_t, o_t and f_t respectively.

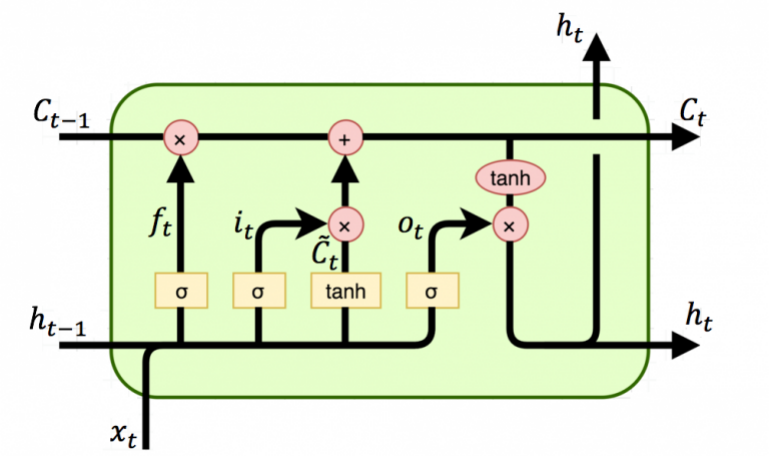

Figure 2.2: LSTM Cell
(Olah, 2015)

The input gate controls the limit of new inputs (x_t) flowing into the RNN layer and determines which value needs to be updated. The output gate controls the threshold of how many times the value in the cell can be used and determines which part of the cell state would be forwarded to the output layer. The forget gate is meant to allow the RNN to remove irrelevant information coming from inputs (x_t) and previous outputs ($h_t - 1$) (Singh and Singh, 2020).

Cho et al. (2014)'s primary goal was to reintegrate the network into an existing SMT system. Therefore, the role of the RNN architecture was to generate phrase pairs from the source data, while the SMT performed the actual translation by computing the conditional probabilities of those phrase pairs. The system was evaluated using BiLingual Evaluation Understudy (BLEU) score, a robust evaluation metric specifically designed to measure translation systems' performance, and it reported a best-performing score of approximately 34%.

The acclaimed Encoder-Decoder model would later be re-implemented by Sutskever et al. (2014). In their system, a model was conceived using RNN networks with multi-layered *LSTM* cells. However, in contrast to Cho et al., the Encoder-Decoder model was used for the entirety of the translation. The solution was tested on an English-to-French translation scenario by training on the publicly available WMT-2014 datasets. The initial baseline result was a BLEU score of 34% using an 80,000-word vocabulary and a left-to-right *beam search* decoder.

Beam search is a search algorithm for picking a target output variable based on the most substantial probability of the next word/character in the sequence. It is called *beam search* because it constructs a search tree which considers different candidates for the most likely element to follow in the sequence. The beam size (a.k.a. beam width) determines the number of tokens considered by the search algorithm. Beam searching outperforms earlier greedier algorithms that always arbitrarily pick the most probable element in the present position without considering a broader context. For example, in an article about dog shelters, the misspelt word 'dor' would be corrected to 'dog' using a greedy algorithm because of the high probability of the word appearing in sequences thus far. However, a beam search could open the possibility of the token being corrected to 'door', which could become the most likely candidate as further information about the sequence is revealed. Therefore, in this rudimentary example, a greedy algorithm would incorrectly score the phrase 'The dor was kept open'. Figure 2.3 illustrates Beam Search computing the probabilities for 'hello world!'.

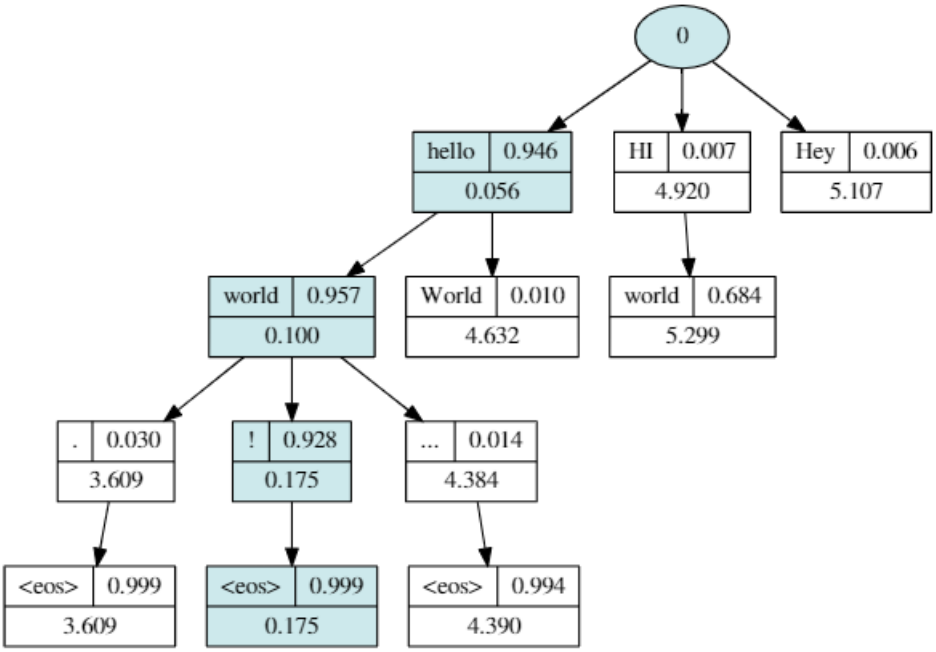

Figure 2.3: Beam Search
(Olah, 2015)

The unoptimised baseline model of Sutskever et al. had exceeded the results of comparable SMTs of that era (Schwenk, 2012). The authors commented that they did not experience any degradation due to long sentences because they reversed the order of the elements in the source language sequences while keeping the same order in the target language sequences. The phenomenon of losing translation accuracy because of sentence length was observed not just by Sutskever et al., and it had caused issues for other researchers in their respective studies (Bahdanau et al., 2014; Cho et al., 2014). In a separate study, Pouget-Abadie et al. (2014) conceived a solution to the issue by segmenting long sentences into smaller phrases, passing each phrase through a translation process, and concatenating the results before yielding the final output. Pouget-Abadie et al. trained their system on the same WMT-2014 datasets as Sutskever et al. and achieved a BLEU score of 20% when evaluated against the *news-test 2014* dataset. The quality of this solution fell short of Sutskever et al. because it translated each phrase in isolation, significantly dampening the fidelity of the output. While the approach of Pouget-Abadie et al. was less accurate than that of Sutskever et al., it was more ro-

bust when dealing with unknown words. In the absence of such solutions, the typical recourse would be to increase the size of the representation vector. However, the memory impact of this approach would be larger than either of the previously described alternatives. The ability to tackle unknown words was a commonly cited advantage of many Encoder-Decoder models over their phrase-based SMT forerunners. Therefore, NMTs were a better approach to generalisation even when working with smaller training datasets which yielded credibility to the argument regarding the successful application of NMTs in critical environments.

So far, all the Encoder-Decoder networks that we mentioned fall under the cap of Seq2Seq. Seq2Seq simply means that a sequence from one domain is to be translated to a sequence of a new domain, which is the basis of all Encoder-Decoder translation models. In 2014, Bahdanau et al. conceived their own version of the Seq2Seq model, also based on the RNN designs adopted by Cho et al. (2014) and Sutskever et al. (2014), but with a special additional adaptation. As evidenced by the previous studies, one of the biggest issues faced by these models was the deterioration of the translation quality as the size of the sentence increased. Bahdanau et al. addressed this problem by creating an extension that allowed the neural network to align and translate words jointly. It worked by performing a soft search on the sequence to identify which positions in the source sentence contained the most relevant information. After identifying these positions, the representation vectors associated with the tokens in those positions were used to predict the target word. All previously generated target words were considered during this prediction phase. The decoder appeared to focus on certain parts of the sequence in practice. The extension was thus called *the attention mechanism*.

The system, dubbed *RNNSearch*, was applied to the same English-to-French translation scenario and thus trained using the same WMT-2014 datasets and evaluated against the *news-test 2014* dataset. The system obtained a BLEU score of 28% when evaluated against the dataset containing sentences with unknown words. The score climbed to 36% once the authors removed the unknown tokens. The results surpassed the long-established SMT solution named *Moses* (Koehn et al., 2007) and superseded the results obtained by Cho et al. (2014) and Sutskever et al. (2014).

In 2017, Vaswani et al. formalised *transformer* architectures. Transformers were an adaptation of Seq2Seq models that worked with the same principles as those elaborated by Bahdanau et al. (2014). The design of transformers mitigated the limitations of LSTM/RNN units in long-sequence applications by using *positional encoding*. A diagram of the transformer architecture can be referred to in Figure 2.4 (Vaswani et al., 2017).

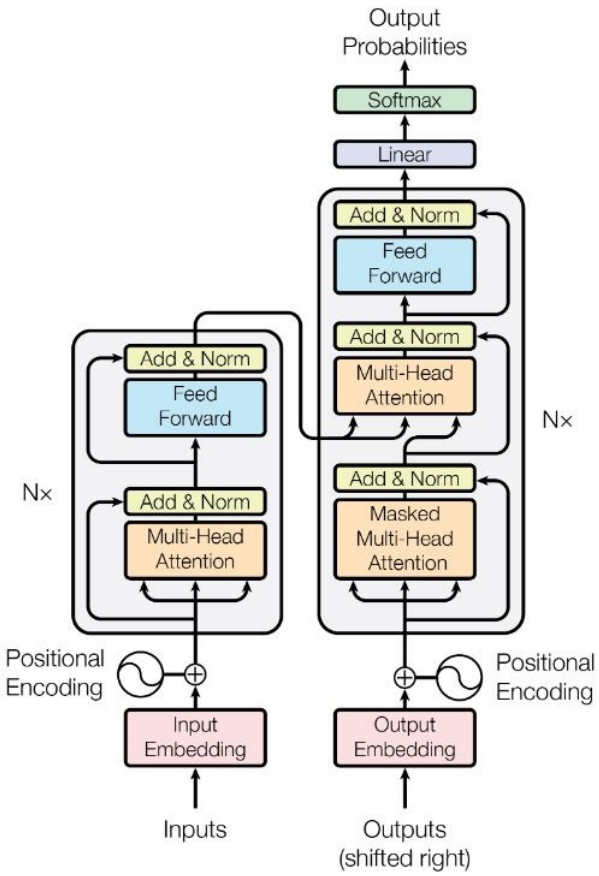

Figure 2.4: Transformer Architecture
(Vaswani et al., 2017)

Positional encoded vectors are element vectors that convey the meaning of the word and context based on its position within a sentence. The input embedding is passed to a *multi-head attention mechanism*. The role of the attention mechanism is to determine which elements of the sequence are most relevant to that sequence. Attention vectors are computed for each word, scoring the relationship between each word. The attention vectors are subsequently fed to a feed-forward network whose role is to simply translate the output of the attention mechanism into something more digestible by an encoder/decoder block. The decoder includes an additional attention block to compute the relationship between each element within the encoded input and the predicted output.

Linear Layers are used to expand/contract the dimensions to the number of elements required in the output sequence. Lastly, a *Softmax Layer* translates the information into a probability distribution that can be interpreted as candidate corrections for the element. The highest candidate can be output as the model's chosen prediction. All tokens are

processed until reaching the end of the sequence. Both the Linear and the Softmax layers are implemented using feed-forward networks.

2.6 | Hybrid Systems

In 2016, Chollampatt et al. came up with one of the first examples of an NMTs system applied in Grammar Error Correction. However, the authors used the neural translator to assist an established SMT model in this study. The researchers noted that neural networks could produce better and more accurate generalised models because they are better suited for capturing and modelling non-linear relationships. The system achieved state-of-the-art results when evaluated against the test sets made available for the Conference on Natural Language Learning (CoNLL) 2014 shared task. CoNLL is a yearly conference that focuses on different themes in natural language processing. For 2013 and 2014, the focus was on Grammar Error Correction.

The combination of SMTs and NMTs together in a supported hybrid approach produced the best results for that time. Following the example of Chollampatt et al., several other studies attempted to build solutions based on similar principles. Rozovskaya and Roth (2011) conducted a research exercise which compared established state-of-the-art systems and benchmarks. Some designs were implemented or replicated, including the top performing system that was identified in CoNLL 2014, a hybrid system (Susanto et al., 2014), an SMT with a discriminative ranking mechanism (Mizumoto and Matsumoto, 2016), a self-authored classifier and a self-authored classifier+SMT hybrid. It was discovered that SMTs are stronger when solving complex errors across long phrases, while classifiers were good at making generalisations about specific error types. Ultimately, the authors championed the adoption of hybrid systems to combine the strengths of classifiers and machine translation techniques.

Hybrid systems continued to feature in the state-of-the-art. By 2018, Grundkiewicz and Junczys-Dowmunt proposed an NMT and SMT hybrid system based on the work of Xie et al. (2016). For this system, the authors implemented the then popularised style of Encoder-Decoder model based on RNNs. Their investigations concluded that a hybrid system with certain architectural considerations such as an attention mechanism could deliver a model that could nearly reach human performance levels. For comparison, they tested out both the SMT and the NMT aspects individually before testing the overall hybrid system, which outperformed both of its constituents individually. When evaluated against the CoNLL 2014 dataset, the system scored 60% precision and 30% recall. The authors also cited 50% M^2 and 56% Generalised Language Evaluation Un-

derstanding (GLEU) scores because participants used both of these evaluation metrics during CoNLL 2014, and the authors could better demonstrate the improvement over the established benchmarks that they had collected.

2.7 | Neural MTs in Grammatical Error Correction

Luong et al. (2015) and Yuan and Briscoe (2016) presented GEC solutions that were entirely built using neural techniques. Both studies delivered RNN implementations based on layers of LSTM cells. The system of Yuan and Briscoe was an EncDec architecture with an adaptation based on the earlier study of Luong et al.. The work of Luong et al. dealt with the issue of Out-of-Vocabulary (OOV) tokens by using an alignment algorithm. OOV tokens are words that do not exist in the vocabularies of the concerned languages. The issue negatively impacts GEC much more than cross-lingual translation. It is an ordinary and necessary practice to invoke a limit on the size of the vocabulary files, primarily to reduce the computational requirements of the model, as the vocabulary sizes are bound to increase proportionally to the volumes of training data. When limiting vocabulary sizes, the more common tokens are prioritised to give the system the best chance of delivering correct translations. Therefore, it stands to reason that the vocabulary of the source language would usually be more extensive in GEC scenarios than in language translation scenarios since, in addition to storing valid tokens, the file would also have to contain a large number of misspelt word tokens.

The algorithm of Luong et al. worked by replacing unknown words with direct translations of their source words that their solution fetched from a dictionary. Of course, this method was specifically geared towards language translation. Yuan and Briscoe adopted a similar approach for GEC. Since the source and target languages were essentially the same, the system could often replace words with themselves. When the authors tested the system against the CoNLL-2014 test set, it scored a $F_{0.5}$ score of 39.9%.

In the same year, Xie et al. (2016) illustrated that an NMT could avoid the OOV vocabulary by using a different type of attention mechanism. Unlike the one implemented for Bahdanau et al. (2014) which operated at a token level, the one of Xie et al. operated at a character level. This adaptation created unorthodox and unexpected results. On the one hand, the model was still capable of correcting long phrases, sometimes even changing token placement into more grammatically consistent formats. On the other hand, the model suffered from new issues such as constructing the target words incorrectly with jumbled characters. Akin to the study of Yuan and Briscoe, this system was also evaluated against the CoNLL2014 shared task datasets but only achieved a $F_{0.5}$

score of 34%.

Based on the foundations laid by Bahdanau et al. (2014), Sennrich et al. (2017) presented *Nematus*, an NMT Encoder-Decoder toolkit with the novel attention mechanism. Nematus was based on DL4MT, an earlier Sequence-to-Sequence model. The difference between the Nematus model and the one of Bahdanau et al. was that the RNN network was designed with Gated Recurrent Units (GRUs) instead of the more commonly adopted LSTMs. Like LSTMs, researchers developed GRU architectures to tackle the vanishing gradient problem. Referring to Figure 2.5, GRUs rely on update gates (z_t) and reset gates (r_t). The gates are two vectors which determine which information the network should pass to the output. Update gates allow the GRU cell to decide how much information forwarded from previous time steps should be sent to future time steps. Meanwhile, reset gates determine how much of the past information should be abandoned and forgotten.

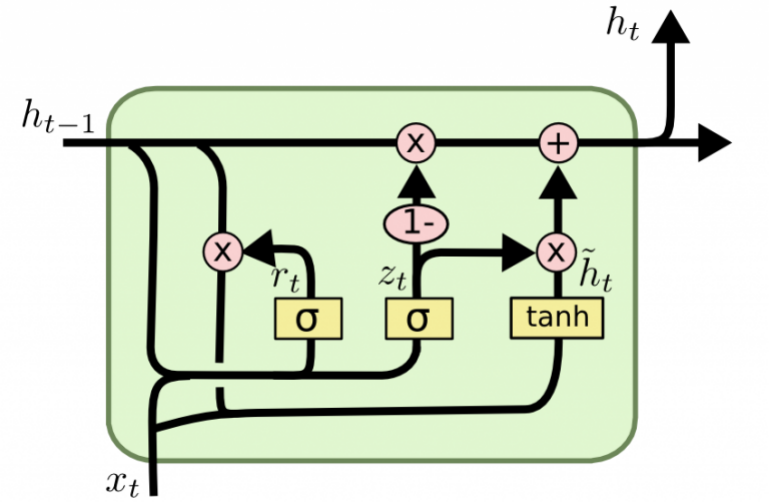

Figure 2.5: GRU Cell
(Olah, 2015)

Nematus was used in several submissions of the Workshop on Statistical Machine Translation 2016 (WMT-2016) shared task and was instrumental in establishing a clear advantage for NMTs in the state-of-the-art. Examples include Junczys-Dowmunt and Birch (2016) who created a system based on Nematus for cross-lingual translation between English, German and French. The system improved on earlier attempts by introducing better preprocessing, including a phase of true-casing. True-casers process texts and repair the letter case errors, such as when the initial letters of sentences are not capitalised or when acronyms are not capitalised. The authors also included a phase of back-translation whereby texts from the target languages were translated into the source

languages using different means and then passed back into the model. The aim of this was to achieve domain adaptation. Domain adaptation simply means that the subject matter of the training data was matched as much as possible to the subject matter of the evaluation sets so that the system performed as strongly as it could. It achieved BLEU scores ranging from 27% to 37% depending on the language pair being translated.

2.8 | Low Resource Environments

By the late 2010s, NMT solutions were already establishing themselves as the state-of-the-art in GEC applications. However, the question remained if NMT solutions were suitable for resource-limited applications. While languages like English and French were excellent proofs of concept for GEC tools, languages like these were quite well equipped with voluminous training data. This advantage was not available for all other languages, which could find themselves in a more demanding position to acquire the data needed for these kinds of systems.

Such was the case with languages like Hindi and Telugu, both of which were used as case studies for the *Sequence-to-sequence text Correction Model for Indic Languages (SCMIL)* (Etoori et al., 2018). In this study, Etoori et al. (2018) created *synthetic datasets* by gathering spelling errors that had a high probability of appearing and introducing noise into a clean corpus. The system was implemented using an Encoder-Decoder setup, supported with LSTM cells. The attention mechanism played a role in alleviating the entire network from additional overheads since the network could spread information throughout the sequence rather than having it encoded into a fixed-length vector. SCMIL was evaluated against Moses. It outperformed Moses by an accuracy margin of approximately 25%, clocking in at 85% and 89% for Hindi and Telugu, respectively. The authors noted that RNNs with GRU units outperformed earlier CNN implementations, though in their case, LSTM units outperformed GRU units.

Similarly to Junczys-Dowmunt and Birch (2016), Sennrich et al. (2016a) used Nematus for translation between English, Czech, Romanian, German and Russian, and ended up citing very similar results (28% to 38% BLEU score). Like Junczys-Dowmunt and Birch, they also performed back-translation. Additionally, the authors used dropout to reduce the issue of the model over-fitting. Dropout works by dropping (removing) certain neurons and their connections from the neural network layers. The idea is to force the network not to overly depend on certain neurons and keep the neural network from over-training for certain scenarios. Both studies employed data synthesis via back-translation.

Náplava and Straka (2019) applied Encoder-Decoder attention models for the data-restricted languages Czech, German and Russian. In each case, the model used synthetic datasets. In the case of Czech, the authors also trained the system on synthetic datasets from different languages and speakers. Two of the datasets were sourced from the essays of Slavic and other foreigners. A third dataset comprised sentences with an entirely different dialect, that of a Romani populace. All the systems significantly outperformed the results of rival systems that the authors tested on the same data. The practice of data synthesis would continue to take different forms as NMTs methods would continue to be applied for GEC.

Around the same period, Junczys-Dowmunt et al. (2018b) published their investigations about low-resource GEC. They re-implemented Nematus into a new system named *Marian* which included most of the original GRU-based RNN implementation. The authors also included several adaptations meant to enhance the system's results in cases of resource scarcity. These were source word corruption, tied embeddings, domain adaptation, error adaptation, edit-weighted Mean Likelihood Estimation (MLE) and pretrained embeddings for transfer learning. Like Etoori et al. (2018), the authors implemented attention mechanisms on the architecture. However, they relied on GRU-based RNN layers in this case. The model achieved a 56% M^2 score.

Flachs et al. (2021) also set out to determine the effectiveness of synthetic datasets when paired with fine-tuning from *gold standard* data. Gold standard data refers to data that has been vetted and corrected by human experts and is therefore considered 100% correct. The authors based their work on that of Náplava and Straka (2019). They concluded that sources such as Wikipedia revisions and datasets like Lang8 yielded the best-performing models across different languages (Czech, Russian, German, and Spanish). Additionally, the authors noted that including adaptations such as the ones recommended by Junczys-Dowmunt et al. (2018b) could be potentially beneficial. This study aimed to confirm the effectiveness of noise synthesis instead of superseding the results established in the state-of-the-art, so the authors did not investigate these aspects further.

2.9 | The BEA-2019 Shared Task

In 2019, a shared task was announced to focus on Grammar Error Correction. The Building Educational Applications 2019 (BEA-2019) Shared task was the spiritual successor to other fruitful shared tasks such as the Helping Our Own (HOO) and CoNLL. The main aim was to create a controlled environment where the authors could evaluate

different GEC solutions under the same conditions (Bryant et al., 2019).

Two determining factors made BEA-2019 stand out. Firstly, there was a revamp of the testing datasets. The original datasets, such as those used for CoNLL were very limited and only contained 50 essays, constituting approximately 1,300 sentences. The subject matter of the texts only spanned two different topics and included the contributions of 25 writers. The new datasets used for BEA-2019 were much more extensive. They consisted of 350 essays, which comprised approximately 4,400 total sentences. The entire span of subject matter included 50 topics authored by no less than 330 different individuals (Bryant et al., 2019). This new dataset was called *WI+LOCNESS* because it was constructed by merging the Write & Improve (WI) corpus with the The Louvain Corpus of Native English Essays (LOCNESS) corpus.

Splitting the task along three distinct tracks was the second factor that made BEA-2019 stand out. These tracks represented different conditions the participants had to adhere to be eligible for submission. Participants could make submissions to a single way or even multiple tracks, and the underlying system could be the same as long as each channel's conditions were honoured.

The first track was named the *Unrestricted Track*. In essence, this was a free-for-all. Any submissions made to this track were only required to honour the essential requirement, i.e. to evaluate against the *WI+LOCNESS* dataset. There were no additional restrictions, and the authors could rely on whatever data they wished, whether privately or publicly available.

The second track was named the *Restricted Track*. This track exemplified previous shared tasks like CoNLL. Participants were restricted to training on a set of annotated learner datasets and were not allowed to use additional datasets for training. The datasets within the *Restricted Track* consisted of only publicly available sources.

The third and last track was the *Low Resource Track*. This track focused on delivering GEC platforms in low-resource environments. Therefore, the datasets allowed in this track were limited, and participants could not use any other datasets during their training. The main goal of this track was to generate interest in developing solutions that could be viable in environments where annotated datasets are extremely limited or do not exist.

In BEA-2019, all participants had to cite their scores in terms of precision, recall and $F_{0.5}$. In practice, all three of these metrics are calculations based on true positives (TP), true negatives (NP), false positives (FP) and false negatives (FN). True positives and true negatives refer to when observations are correctly predicted as belonging or not belonging to the classification. In GEC, these would be instances when incorrect words are corrected (true negatives) and when correct words are left as is (true positives). False

positives and false negatives refer to when those mentioned earlier are classified incorrectly. In GEC, these would be instances when an incorrect word is left uncorrected (false positives) and when a correct word is erroneously changed (false negatives).

The most important metrics are precision and recall. Precision (see Equation 2.3) measures the exactness of the model and indicates if the tool is amending the correct tokens. Recall (see Equation 2.4) measures the completeness of the model and indicates how much of the intended sample space is covered by the spell checker. $F_{0.5}$ (see Equation 2.5) is a computation of precision and recall where precision is given greater importance than recall.

$$Precision = \frac{TP}{TP + FP} \tag{2.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

$$F_{0.5} = 1.25 * \frac{Precision * Recall}{0.25 * Precision + Recall} \tag{2.5}$$

All scores cited by participants had to be generated using the ERRor ANnotation Toolkit (ERRANT) scorer. This scorer was an advanced version of the earlier M^2 scorer, which could deliver more detailed feedback, including error types Bryant et al. (2019).

This shared task, in particular, was the most recent shared task to tackle low-resource GEC and publish its results. Bryant et al. states that even though participants were not forced to use one paradigm over the other, two-thirds of all the participants ended up using transformer-based NMT methods, drawing particular inspiration from the work of Vaswani et al. (2017). In the case of the *Low Resource Track* in particular, the split was even more biased. There were many submissions but a publication did not accompany every submission. Several were basing their solutions on that of Junczys-Dowmunt et al. (2018b) who had achieved state-of-the-art results in the previous CoNLL shared task.

Unsurprisingly, the top ranking system in the *Low Resource Track* was that of Junczys-Dowmunt and Grundkiewicz (2018) with their submission titled, Microsoft and University of Edinburgh (MS-UEdin). The system used both an SMT and an NMT. The NMT was implemented using Marian NMT, which was already described thoroughly in Junczys-Dowmunt et al. (2018b). Several top contending systems, such as Kakao and Brain, did not have an accompanying paper. Meanwhile, others that had failed to submit in time for the publication of the results (Bryant et al., 2019) still conceived effective systems that they evaluated against the same benchmarks, such as Kaneko et al. (2020)

who built a low resource system based on BERT models. Bidirectional Encoder Representations from Transformers (BERT) is another EncDec implementation that uses an attention mechanism. Kaneko et al. (2020) fine-tuned an existing BERT language model to pretrain their GEC tool, which was also incidentally based on a transformer. They concluded that using the pretrained BERT model increased the quality of their model across the correction of all error types.

2.10 | Summary

In this chapter, we embarked on a historical rundown of Grammar Error Correction. We examined how they started as simple rule-based systems and eventually transitioned into complex machine translation architectures based on either statistical or neural methods. Ultimately, the state-of-the-art shifted from SMTs to NMTs due to NMTs requiring fewer resources to implement and yet still achieving similar degrees of success.

The world of Grammar Error Correction changed entirely with the introduction of Encoder-Decoder models. These models consisted of an encoder tasked with translating words from the source language into numerical vector representations and a decoder that decoded these vectors back into words belonging to the target language. By organising GRU/LSTM cells within an RNN architecture, the entire context of long sequences could be stored and repurposed for future input into the same networks during feedback loops. These kinds of NMTs were christened as *Sequence-to-Sequence models* for how they ferry sequences between the encoder and the decoder.

Eventually, researchers introduced an innovative add-on called the *attention mechanism* to Seq2Seq models. This mechanism significantly decreased the hardware resource requirements of the Seq2Seq architecture and made the pipeline faster. Models with an attention mechanism eventually branched into a new category called *transformers*. Due to their particular characteristics, transformers became widespread in low-resource environments. Their implementation was often coupled with adaptations to counteract the limitations of resource scarcity.

The most notable adaptations include synthesising errors via the purposeful artificial noising of source datasets. These allowed researchers to enlarge their training data by creating grammatically incorrect sets from valid texts, effectively creating training observations from thin air. Another strong adaptation was tied embeddings which involved relating the embedded layers of the source and the output together. There was transfer learning, whereby the translation model would base its initial weighting on a

pretrained language model such as a BERT model.

The relatively recent BEA-2019 shared task fostered a community of researchers who vied to develop the strongest low resource GEC. The fruits of the BEA-2019 shared task culminated in many systems primarily based on transformer architectures and included adaptations like the ones mentioned. Presently, the submissions of the BEA-2019 shared task occupy the top ranks within the state-of-the-art, making this particular shared task an excellent point of reference for developing and evaluating a new GEC tool for Maltese.

3

Implementation

This chapter outlines the method implemented to achieve the aim of the study. It includes all the algorithms, datasets and technologies that played a role in constructing the final models. In research and development, there were several instances when experiments led to unfortunate dead-ends. While these efforts bore no fruit towards the final artefact, they were still necessary steps towards formalising the correct process that eventually yielded complete results. The failed experiments have been included in the documentation for these reasons.

The chapter is sectioned as follows. Section 3.1 outlines the motives behind the experimentation process from beginning to end. Section 3.2 describes the various technologies used during the implementation. Section 3.3 details the process undertaken to replicate existing studies from the state-of-the-art. Section 3.4 describes how we executed our own vision for a Maltese GEC system. Section 3.5 discusses the adopted experimentation strategy, including the calibration of the many hyper-parameters supported by the Marian NMT toolkit. Section 3.6 contains a summary of the entire chapter.

3.1 | Methodology

3.1.1 | Justification for the use of NMTs

In Chapter 2, the literature establishes neural methods as the strongest contenders for creating a GEC system. Both statistical and neural machine translation methods rely on high-quality training data. If we chose to use SMTs, much larger datasets would have been required to create the GEC solution, which would have been a problem because Maltese does not provide the best-provisioned environments. One of our biggest challenges was finding enough data for training. Furthermore, the availability of hard-

ware resources (such as Tensor Processing Units (TPUs) or Graphical Processing Units (GPUs)) for training was an additional challenge that we faced while operating from a low-resource environment.

We chose to use NMTs instead. As evidenced by the work of (Junczys-Dowmunt et al., 2018b), modern neural techniques outperform statistical techniques when operating with smaller volumes of data. It holds for both the accuracy obtained and the hardware requirements during training. NMTs have a higher rate of improvement with smaller datasets than SMTs, and they are better equipped to handle OOV issues. Since we were using smaller datasets, we were bound to encounter words that were unavailable in our vocabularies. The absence of a word from the training datasets means there is no way for the system to compute a potential correction.

Nevertheless, this is a less detrimental problem in the case of NMTs. A neural network architecture allows for handling never-before-seen words as their input vector representations can still be propagated through the neurons as usual. This advantage further emboldens us to champion NMTs as the best machine translation alternative for Maltese.

While we intended to use as much data as possible, we knew we would not have the volumes typical of other GEC projects. A neural solution offsets the need for large amounts of data by imposing broader contextual awareness in its design. We decided to implement models based on RNN-based Seq2Seq architectures. These are better suited to handle large input sequences and incorporate them into the weighting mechanism. Furthermore, we built transformer models. These models are akin to Seq2Seq but also include the attention mechanism. This mechanism enables the model to recall its input context and factor the token position with the phrase semantics.

3.1.2 | Considerations for Low-Resource Environments

Several considerations have been proven to work well in low-resource environments to improve the model's accuracy or reduce the hardware requirements. Junczys-Dowmunt et al. (2018b) demonstrated how adaptations such as source word corruption, domain adaptation, error adaptation, tied embeddings and pretraining could improve the results of a neural machine translator in a low-resource setting. We attempted to implement all of these adaptations in our solution to varying degrees of success.

3.1.2.1 | Source Word Corruption

Source word corruption is a noising technique that purposely drops words from the source text. It achieves an effect mimicking when users erroneously miss out on a word while

writing a sentence. More critically, it allows the network to train that it does not overly rely on certain words during error correction. For example, if the network trains to correct 'in the of time' to 'in the nick of time', it could be better adapted to correct a broader range of errors that include misspelling 'nick' but also instances when the word is missed entirely or incorrectly substituted with another word. For low-resource environments, this adaptation could help the GEC model perform beyond the limitations incurred due to a lack of training data since it would allow the network to correct errors that did not necessarily feature in the training sources.

Knowing the distinction between this type of corruption and source dropout that occurs at a network level is essential. On a network level, one can apply dropout mechanisms on the source word inputs that are part of the input layer of the RNN network. This type of dropout does not involve corruption of the source files and works according to a probability setting. The setting indicates the random chance of dropping nodes from the network. The most significant difference is that the dropped neurons can change from one epoch to another. In contrast, when corruption is introduced to the source data, the dropped words are eliminated for the training duration.

For our solution, we included source (neuron) dropout in all our experiments because, in the original study of Junczys-Dowmunt et al. (2018b), this technique was implemented in the baseline. We then used our error synthesis tool (explained in Subsection 3.4.2.1) to perform source word dropout on the datasets.

3.1.2.2 | Domain Adaptation

Domain adaptation is the practice of creating a model that is as domain agnostic as possible (unless one is purposely aiming for the GEC tool to be biased). In technical terms, this can be achieved by doing two things:

1. Use text sources from as many domains as possible such as news articles, blog posts, online magazines and chat logs.

2. Ensure that all of these corpora are approximately the same size.

For low-resource environments, it might be difficult to have large enough volumes of data. As a countermeasure, one could manually duplicate rows from different corpora until they reach a similar size. In our solution, we performed an initial training phase on a dataset comprising 4000 sentences. We then merged this dataset with two more datasets from different domains to increase the size of our training data by three times.

3.1.2.3 | Error Adaptation

The relative equality between different corpora as described in Subsection 3.1.2.2 also applies to the error rate. In order to avoid creating bias in the model, the error rates of different corpora should ideally be similar. In low-resource environments, this is achieved using error synthesis, which means that errors are introduced into the corpora on purpose. There are several strategies for doing this, including basic methods like introducing errors of omission, insertion and substitution. Other elaborate methods include introducing keyboard proximity, domain-specific, or even phonetic errors using Soundex. The percentage of errors in different corpora should be approximately equivalent.

In our solution, we used our error synthesiser tool (explained in Subsection 3.4.2.1) to introduce errors into our two new training sets.

3.1.2.4 | Tied Embeddings

Tied embeddings were first introduced by Press and Wolf (2017) to improve perplexity. On a technical level, it involves tying the weights of the word embeddings to the network soft-max layers. *Word embeddings* are vectors that mathematically describe word tokens. Embeddings can be tied simultaneously on both the encoder and decoder sides.

We ran our initial models during our experiments with embeddings tied only on the decoder side. This measure was essential for any Marian-implemented architecture to eventually output sequences compatible with new vocabularies. As an additional adaptation, we tied them across all layers, which immediately forced us to harmonise the source and target vocabularies into one homogeneous vocabulary.

Models with tied embeddings typically produce better results in low-resource environments than those without the adaptation. Additionally, tied embeddings reduce the number of parameters required for the neural network, thus creating a smaller model with fewer negative implications on the hardware.

3.1.2.5 | Pretraining

Pretraining involves using a trained model as the starting point for a new training exercise. In low-resource environments, this could increase the accuracy of the resulting models. There are different kinds of pretraining such as applying a trained model from one domain to a different domain or applying a trained model from one language to another.

We trained on top of two separate BERT models for our study. These models are better explained in Subsection 3.2.1.2. With this method, the initial weights of the network are influenced by the additional pretrained language model.

3.1.3 | Maltese as a Low-Resource GEC Scenario

Maltese is a prime example of a low-resource language. With a native-speaking population of roughly half a million people, Maltese is one of the least commonly spoken languages in the world. Like other uncommon languages, Maltese suffers from a lack of resources for several reasons:

- As a bi-lingual country, the population is likely to communicate on online platforms using English rather than Maltese.
- Many people do not use Maltese keyboard fonts and write Maltese alphabetical letters such as *ċ*, *ġ*, *ħ* and *ż* incorrectly. These are often replaced with the letters *c*, *g*, *h* and *z* respectively. The letter *c* does not exist in the Maltese alphabet, while the others all exist but are distinct from the aforementioned characters (*ġ*, *ħ* and *ż*).
- A general lack of Maltese online content has led to a decline in speakers' knowledge of orthography. Furthermore, much of the Maltese content has numerous grammatical and spelling errors. Reading conventional books is a declining trend, so exposure to Maltese mostly comes from social media, which is wrought with spelling and grammar errors.
- Concerning storing written texts in electronic format, Maltese has lagged compared to other languages and has yet to catch up.

Fortunately, some characteristics of Maltese writing simplify the challenges a GEC implementation faces.

- It follows a left-to-right writing pattern.
- Aside from removing *c* and adding the characters *ġ*, *ħ* and *ż* - the Maltese character set is identical to the Latin character set.
- The language is read and spoken phonetically, which potentially minimises the variance between a correctly and incorrectly spelt word.

For these reasons, Maltese is a great candidate for evaluating the efficacy of a low-resource GEC solution.

3.2 | Technologies

3.2.1 | Toolkits & Models

There are numerous toolkits available for language modelling. Some researchers develop models using fundamental deep learning frameworks such as PyTorch[1] and TensorFlow[2]. These powerful machine-learning libraries boast several options for developing powerful AI models. Among those models, Seq2Seq models and transformer architectures like those described earlier could be developed.

In the case of NLP, more focused options are available to choose from, like HuggingFace[3], which is an extensive library of NLP models, including tokenisers, lemmatisers, dependency parsers, sentiment analysers and even language translators. HuggingFace lets developers download pretrained models from a vast cloud-based library and fine-tune them.

Lastly, there are also available many toolkits that are geared for NMT tasks, such as Open NMT[4] and Marian NMT. Both of these are NMT toolkits. Open NMT is an open-source toolkit with both an implementation in PyTorch and TensorFlow. Marian NMT is owned by the University of Edinburgh (but it is still publicly available), and it is built in C++. Toolkits like these have the benefit of being specialised for dealing with NMT projects and their requirements. Most often, the toolkits are packaged with external bundles of libraries and resources that are commonly used in NMT.

3.2.1.1 | Marian NMT

We decided to use the Marian NMT toolkit for this project. Marian NMT[5] provides developers with a powerful API suite that is specifically made for building translation pipelines using various Encoder-Decoder models, including Seq2Seq and transformers. Chief among its supported transformer implementations are those of Vaswani et al. (2017) and of Sennrich et al. (2017). It also allows for fine-tuning these models by calibrating learning rates, optimiser parameters, network depth, cost functions and even the type of network cells being used (such as LSTM or GRU). There are several reasons why Marian is the preferred choice for this study.

As described in Section 3.2.1, there were other NMT toolkits or machine learning frameworks to choose from. However, studies often did not make their code publicly

[1] https://pytorch.org/ (accessed: 2022-03-12)
[2] https://www.tensorflow.org/ (accessed: 2022-03-13)
[3] https://huggingface.co/ (accessed: 2022-03-13)
[4] https://opennmt.net/ (accessed: 2022-03-18)
[5] https://marian-nmt.github.io/ (accessed: 2022-04-03)

available and were impossible to replicate and verify their results. Marian NMT was the only toolkit that offered extensive options for low-resource adaptations. Other toolkits like Open NMT had their own features to compete with but were scarcely ever chosen in past studies and thus did not achieve results that were comparable to those obtained by Marian NMT. Due to its adequacy for low-resource environments, its uncontested position in the state-of-the-art and its public availability, Marian NMT was the natural choice for implementing our system.

Marian was used by Junczys-Dowmunt et al. (2018b) in their detailed investigation into NMT-based GEC in low-resource environments. The authors continued improving their results with future studies, increasing the effectiveness of the toolkit's models and the efficiency of the training process. The Marian system was used to produce significant competitive benchmarks in the state-of-the-art and eventually claimed the top spot in the BEA-2019 shared task. It was the best-performing system for the Low-Resource track (Grundkiewicz et al., 2019). As of the writing of this dissertation, the benchmark set by the MS-UEdin submission for the Low-Resource track in the BEA-2019 shared task occupies the top spot in the shortlist. As discussed in Chapter 2, MS-UEdin was based on Marian. Therefore, this is the top-performing toolkit available in the state-of-the-art when it comes to NMT.

Marian is written in Python but uses libraries built in C++. Originally, the initial version of Marian was written purely in C++. However, the developers eventually migrated some elements to Python due to its simple syntax, modern compatibility and widespread use in machine learning applications. Python supports the ability to interface with libraries that are built in different languages. For Marian, a lot of the aspects related to design, configuration and hyper-parameter calibration were moved to Python. Meanwhile, the more time-critical aspects of the toolkit such as the actual implementations of neural networks and attention mechanisms were kept in pre-built C++ libraries. C++ runs much faster than Python and is better suited for high-performance computations. Additionally, Marian exposes command line arguments that can be interacted with using shell scripts. For this study, we relied primarily on these command line arguments to interact with Marian NMT.

The apparent bottleneck in machine learning systems is RAM and processor power. It cannot be understated that the choice of programming language still plays a significant role, especially when considering the large volume of iterations, cost computations and weighting operations needed to train an RNN-based Encoder-Decoder network. In low-resource environments, any advantage, no matter how small, can leave a substantial impact when considering the limitations of available hardware. The relevance of this factor becomes more pronounced when we consider that the shortcomings of the

data might be counteracted by increasing the complexity of the model.

Despite Marian's evident success and potential, other submissions that were shortlisted in the BEA-2019 shared task were studied for comparison purposes. Unfortunately, the source code was often unavailable. Those solutions that shared their source code were typically lower on the shortlist and did not provide strong enough results to compete with Marian's. Another characteristic of the studies conducted by Junczys-Dowmunt et al. (2018b) and Grundkiewicz et al. (2019) was that the translation process was completely unsupervised. A considerable challenge or resource scarcity was the improbability of discovering annotated datasets. This issue eliminated the potential for supervised machine learning solutions as we lacked the necessary parameters for training. While Maltese has undergone annotation exercises before, these exercises were geared towards the building of PoS taggers and dependency parsers - neither of which would prove much use in our GEC scenario. Unsupervised methods allowed us to use unlabelled data sources which were more available.

There were available Marian implementations on HuggingFace that could be harnessed using Python. However, we decided to rely on the official Marian distribution, which could be interacted with purely using its native command line interface. Since an essential phase of this study was to replicate the results of an existing system as faithfully as possible, we needed to use the same methods when possible. Marian models available on HuggingFace did not feature the latest stable version. Additionally, the owners of the Marian system offered online support to people using the official Marian NMT toolkit but not those using an off-shoot from HuggingFace.

3.2.1.2 | BERT Models for Pretraining

According to Junczys-Dowmunt et al. (2018b), pretraining from a monolingual model could improve the result of GEC training. Kaneko et al. (2020) successfully demonstrated that it was possible to use pretrained BERT models as a training base for GEC systems. The study, like that of Grundkiewicz et al. (2019), was also forwarded as a submission to the BEA-2019 shared task. At the time of the implementation stage of this project, there were no publicly available pretrained models for Maltese. However, we managed to contact a researcher conducting a separate study into creating a BERT language model for the Maltese language (Micallef et al., 2022). The two models developed by Micallef et al. are *BERTu*[6] and *mBERTu*[7]. The former is a monolingual model trained on a newer version of the Korpus Malti. At the same time, the latter takes the

[6] `http://huggingface.co/MLRS/BERTu/` (accessed: 2022-05-11)
[7] `http://huggingface.co/MLRS/mBERTu/` (accessed: 2022-05-11)

popular mBERT[8] and further pretrains it with data from the Korpus Malti to produce a new multilingual model that includes Maltese. We authored a specialised script to convert these models into a Marian-readable format. Both of these models are used during experimentation.

3.2.2 | Development Tools

We stayed faithful to the recommendations of Marian's documentation and opted to build our solution on a Linux operating system. We did not have the hardware to support Encoder-Decoder models on our local computers, and cloud platforms like AWS or Azure would have been too costly to maintain for the entire duration of the study. Therefore, we decided to implement the solution on Google Colab.

3.2.2.1 | Google Colab

Google Colab is a specialised platform designed to assist research efforts by providing on-demand computing resources. Code is written in a Python notebook, a document separated into code blocks that could be executed individually or in sequence. Upon executing a code block, Colab connects the user with an available virtual machine that can execute the operation. The platform also supports the execution of bash scripts and shell commands which is ideal for interacting with Marian.

3.2.2.2 | Hardware

Colab offered three types of run-time environments, each using a different kind of processor (CPU/TPU/GPU). Colab typically made available one type of CPU or one of two types of GPU based on circumstance. Table 3.1 presents the specifications for CPUs and GPUs.

In the case of TPUs (Table 3.2), there was a much more comprehensive list of potential allocations, but like GPUs, they were also subject to availability. Tensor Processing Units (TPUs) were designed by Google to work efficiently with TensorFlow. TPUs were not put to their maximum potential for this study. However, the additional cores and memory were nevertheless beneficial for faster training.

Both GPUs and TPUs were used during experimentation when creating a model was required. However, it only took a few hours to exceed the usage quota set by Google, and therefore we often deferred to CPU training in these circumstances. The challenge of being able to train with CPUs only was also a common issue faced in these

[8]Multilingual BERT consists of 104 languages but excludes Maltese.

Table 3.1: Google Colab Environment Specifications (CPUs & GPUs)

Spec	CPU	GPU
Model	Intel(R) Xeon(R)	NVidia K80 / T4
Processor	2.30GHz	0.82GHz / 1.59GHz
Performance	10 - 22 GFLOPS	4.1 TFLOPS / 8.1 TFLOPS
Cores	2	2
Family	Haswell	Tesla
Memory	12Gb	12Gb
Disk	25Gb	358Gb

Table 3.2: Google Colab Environment Specifications (TPUs)

Type	Cores	Memory
v2-8	8	64 GiB
v2-32	32	256 GiB
v2-128	128	1 TiB
v2-256	256	2 TiB
v2-512	512	4 TiB
v3-8	8	128 GiB
v3-32	32	512 GiB
v3-64	64	1 TiB
v3-128	128	2 TiB
v3-256	256	4 TiB
v3-512	512	8 TiB
v3-1024	1024	16 TiB
v3-2048	2048	32 TiB

types of environments. Therefore, the limitation on Google Colab created a challenge to assemble a model that could be trained using CPUs only if there were no other options.

3.3 | Replication

As discussed in previous sections, we focused on two alternate solutions, both of which were submissions to the BEA-2019 shared task and achieved strong results. Before proceeding to build our solution, we needed to verify the results achieved by Grundkiewicz et al. (2019) and Kaneko et al. (2020). We, therefore, attempted to replicate both systems. Fortunately, both studies made their systems available on Git. Since both studies were aimed toward submission to BEA-2019, they used the same datasets for training. A common requirement for the Low-Resource tack was to rely on the WI-LOCNESS corpora (Bryant et al., 2019).

Note that we are also referring to Grundkiewicz et al. (2019) because it happens to

be the study which focused on setting a benchmark for the Low-Resource track in the BEA-2019 task and also made the code and data available to the public. The studies by Junczys-Dowmunt et al. (2018b) and Grundkiewicz et al. (2019) were individual efforts that made use of the same Marian NMT toolkit but either focused on solving a different problem (Junczys-Dowmunt et al., 2018b) or had not made the code available as of writing this dissertation (Grundkiewicz et al., 2019).

3.3.1 | Marian NMT GEC

For the solution of Grundkiewicz et al. (2019), the Marian code base was cloned from a Git repository. The code needed to be built using CMake so that the native C++ compiler could read it. *CMake* is a cross-platform tool that uses configuration files to generate native build tool files. CMake is entirely platform-independent, so the files are generated specifically for the machine's compiler. However, on Google Colab, there is a mismatch between the native CMake compiler and the Marian NMT code files. The mismatch occurs because the machines used by Colab are running using a slightly older version of the tool. CMake had to be manually uninstalled from the Colab instance to get the build working, and a newer version had to be reinstalled directly from the source. Additionally, the build could only finish if executed using the CPU only and purposely refraining from deferring the task to a GPU. After ensuring that these steps were taken, Marian would build successfully.

Subsequently, we downloaded the trained models and tested the decoding script included in the repository, which reproduced the results cited in the paper. For good measure, the training process for the Low-Resource track was also tested for a small number of epochs to ensure that it was working correctly.

3.3.2 | Fairseq GEC

Like the previous replication exercise, the system of Kaneko et al. (2020) was cloned from a publicly available repository. However, in the case of the study, it was far more strenuous to arrive at a functional replication since the repository was left rather unmaintained. The solution was built using Fairseq[9], but most of the method calls were outdated and failed upon execution. An exercise was undertaken to repair the outdated files, which led to fixing six separate code modules that brought the solution up to speed. After this process, the BERT model could be loaded and tested correctly. The results matched those cited in the paper.

[9]https://fairseq.readthedocs.io/en/latest/ (accessed: 2022-02-13)

3.3.3 | Testing for Maltese

Before implementing, we tested out both the Marian and the Fairseq solutions on Maltese datasets. The original plan was to compare the results of the two baseline models to determine which Encoder-Decoder architecture was the strongest. While training Marian on Maltese data was successful, the same exercise did not yield fruitful results for the Fairseq solution (the one prescribed by Kaneko et al. (2020)). The base BERT model had been pretrained on an English corpus and attempting to retrain using Maltese data resulted in architectural mismatches. The issue might have been related to the model being trained using outdated code. The authors did not indicate which version of Fairseq they were using.

Nevertheless, we decided to continue development using the Marian toolkit. Fortunately, the toolkit makes available different Encoder-Decoder architectures, so we still managed to perform a round of comparative testing before committing to a specific type of implementation. While the option of Kaneko et al. was no longer available, this did not take away from the inclusion of pretrained BERT models during the pretraining phase of the experimentation.

3.4 | Execution

3.4.1 | System Schematics

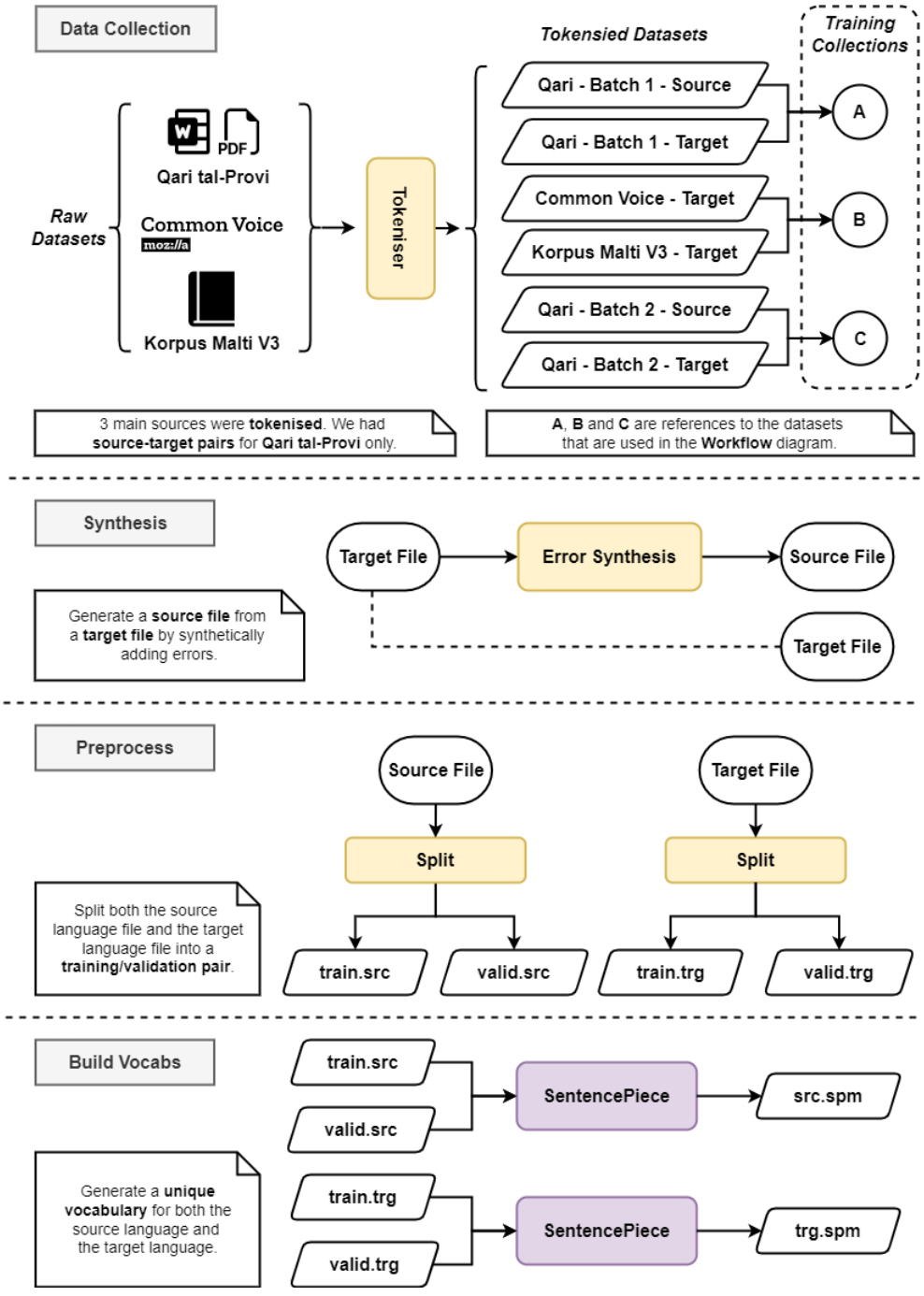

Figure 3.1: Maltese GEC Sub-processes

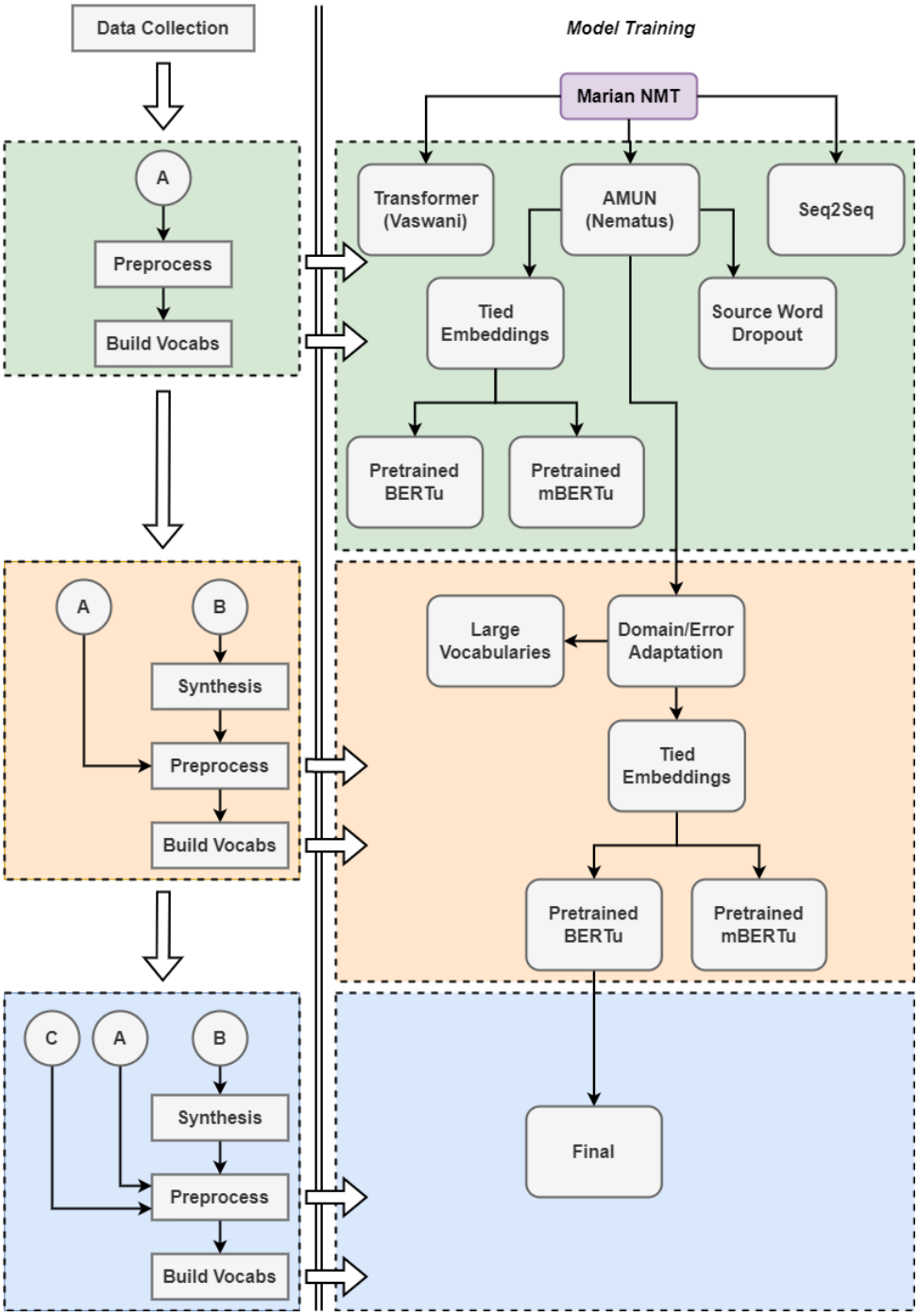

Figure 3.2: Maltese GEC Workflow

Figure 3.1 and Figure 3.2 are schematic diagrams that describe the completed system. Figure 3.1 is a drill-down of some of the important sub-processes involved in the full workflow described in 3.2. There are four major sub-processes.

Data Collection is the phase where we retrieve the data sources, tokenise them and split them into source and target files. The source files consist of the raw input with spelling mistakes, and the target files consist of their corrected equivalents. For clarity, we split the datasets into three collections A, B and C in the diagram.

- Collection A consists of the first batch of the *Qari tal-Provi* dataset (see Subsection 3.4.2.2) that was used to train the baseline models as well as test out *Source Word Corruption*.
- Collection B consists of the Common Voice and MLRS datasets (see Subsection 3.4.2.3 and Subsection 3.4.2.4). These had no source file counterparts and required that those files be synthesised from scratch. This collection was used together with Collection A from the *Domain and Error Adaptation* phase onward.
- Collection C consists of the second batch of the *Qari ta-Provi* dataset (see Subsection 3.4.2.2) that was kept as an evaluation set. This collection was used with both Collections A and B during the generation of the *Final Model*.

Synthesis depicts how a target file is passed through the synthesis tool to create a source file. We used this process on our Common Voice and MLRS datasets (see Subsection 3.4.2.1). *Preprocess* depicts how the source and target files were split into a training and validation file pair. *Build Vocabs* is the step taken to generate vocabularies from the source files (see Subsection 3.4.3.2).

Figure 3.2 depicts the experimentation strategy. The differently shaded areas relate to a phase of experimentation with a collection of datasets used for training and building vocabularies. The experiments are discussed in further detail in Chapter 4.

3.4.2 | Data Collection & Curation

The first implementation phase was to prepare a training corpus for the system. Neural translation systems train by processing parallel string pairs. One element would be the source language, while the other would be the target language. The encoder and decoder networks would be independently trained to understand a concept (a string) represented using context vectors. The encoder handles mapping the source text to these context vector representations, while the decoder handles connecting the representations to the target text. Powerful Encoder-Decoder models manage to translate sentences from a source language to a target language irrespective of the grammatical differences, such as the placement of determiners and word inflexions of verb conjugations. So long as the vector representation is captured accurately enough, the sentences

on either side of the translation can be as distinct as required. They could even be of varying lengths. The same concept applies in our scenario, with the only quirk being that both the input and the target languages are Maltese. However, the input language has grammatical errors while the target language is grammatically correct.

3.4.2.1 | Error Synthesis

The problem with our Maltese resources is that many incorrect texts do not have corrected counterparts. The only natural way to resolve this would have been to recruit human checkers so they could provide us with corrections. Recruiting human checkers was not possible or feasible within the project's time frame. Nevertheless, we could work in the opposite direction and generate errors based on grammatically correct text using error synthesis. We developed an error synthesis tool that could create noise in a token list. The tool was heavily inspired by the work of Náplava and Straka (2019), who also applied their rule-based synthesiser to low-resource languages such as Czech and Russian. They included several synthesis techniques such as character omissions, insertions and substitutions. Our tool also had many different synthesis strategies and assumed a standard 48-key Maltese keyboard (or font set). The complete list of synthesis strategies was as follows:

- Errors of Omission - Characters are removed from a word.
- Errors of Transposition - Two adjacent characters are substituted.
- Errors of Insertion - Characters are added to a word.
- Keyboard Proximity errors - Characters are erroneously typed instead of other characters because of their closeness on the keyboard.
- Duplicate Character errors - Characters are erroneously typed in twice.
- Silent Letter errors - When the writer mistakenly omits a silent letter from a word.
- No Font errors - When the writer is not using Maltese fonts on their keyboard.
- Common Article errors - When the writer makes a mistake in the article.
- Corruption Noising - Drops a token from a sentence sequence.

All of the strategies employed by the tool except for corruption noising are applied using random seeds. The tool could be used with a simple method call that accepted the type of synthesis strategy as a parameter along with a sentence seed, a token seed

and a character seed. All three seeds represented percentages of noising. For example, if we passed a value of 0.1 to each parameter, the tool would apply the noising strategy to 10% of the characters in 10% of the words in 10% of the sentences. In order to preserve the conditions during testing, we ran the synthesiser once and kept the synthesised files throughout the experimentation process.

We first implemented the most straightforward strategies: the application of typos based on keyboard proximity and simple insertion, omission, and transposition errors. These errors are relatively easy to identify in the source data files. Tokens with more characters than expected had undergone an insertion such as the word 'armdata' instead of 'armata' (meaning 'army'). Deletions create the opposite effect, for example, the word 'uwa' instead of 'huwa', which is the Maltese masculine form of the word 'is'. Transpositions happen when two characters have switched positions, like the word 'tadr' instead of 'tard', which means 'late'. Keyboard proximity errors might be harder to notice. When applying these errors, the synthesiser calculates a random chance distribution of the keyboard letters surrounding the intended letter. Afterwards, it randomly replaces the intended letter with one of the options in the distribution while preserving the case. Note that since we are using Maltese keyboards, keys such as '[', ']' and '\' are replaced with Maltese letters. An example of one such error from the data files is 'sllura' instead of 'allura', which is the Maltese word for 'therefore'. In this example, the 'a' was assigned to an 's'. It had an equal random chance of being assigned to a 'q', 'w' or 'z', the closest characters on the keyboard.

We progressively started to include more specialised strategies to mimic some of the more common challenges faced in Maltese spelling. The *No Font* synthesis strategy assumed that the typist would not be using any Maltese fonts, so it would replace the Maltese characters *ċ*, *ġ*, *ħ* and *ż* with *c*, *g*, *h* and *z*. In error adaptation, one should aim to mimic the error rates of the original data files, which in our case were a set of corrected document pairs (see Subsection 3.4.2.2). None of the source files had Maltese fonts, so we decided to apply this synthesis strategy to the entirety of our remaining data files. Examples of these instances include words like 'ggant', 'ghalfejn', 'mizura' and 'addocc', which should have been written as 'ġgant', 'għalfejn', 'miżura' and 'addoċċ' instead. We included the *Silent Letter* strategy so that the tool could remove the silent letters *għ* and *h* entirely, as these were also often missed by writers.

We implemented the *Common Article* strategy to simulate when writers would make errors in Maltese articles. In Maltese, the prefixed vowel of the article should drop if there is a vowel at the end of the preceding word (barring some exceptions). In Maltese, these articles are sometimes miswritten, either not dropping the vowel or dropping it unnecessarily. For example, the phrase 'aqra ir-rapport' should be written as 'aqra r-

rapport' due to the proximity between the last vowel in the word 'aqra' and the first vowel in the natural article 'ir-'.

Lastly, we developed the *Corruption Noising* method, which would delete a word from the string of text. This strategy was included in preparation for the *Source Word Corruption* adaptation experiment. This noising strategy is the only technique that relies on a non-random pattern of synthesis whereby the tool removes the N^{th} token as defined in the method call. In our case, we applied corruption to remove the tenth token in each sentence. In the original study, Junczys-Dowmunt et al. (2018b), the rate equated to removing the fifth token in each sentence. However, since we were dealing with a much smaller pool of data, we decided to half the rate of dropout to remove only the tenth token. The datasets used in the original study had 26,000,000 tokens while ours totalled to only 87,000 tokens. This number coincides with a standard starting rate for source word corruption.

3.4.2.2 | Dataset from *Qari tal-Provi* Diploma Course

One of the primary challenges in a low-resource scenario is having data which is humanly curated. In the case of this project, this would entail having a dataset with text including errors and text including the corrected version. Although such data is not publicly available, we were in a position to obtain such a dataset by contacting the course coordinator for the Diploma in *Qari tal-Provi* (Proof Reading) for Maltese. Such an exercise had already been done previously, resulting in an initial batch which contained Microsoft Word documents (Vella, 2015). The documents contained the corrections through *Track Changes*, and all the changes and accepted changes were logged in the document's XML markup.

To extract valuable data from these files, we built a special parser to explore the XML markup of each document and iteratively mine for each string of text and any associated corrections. The source and target strings were created for each string by keeping the original and applying the amendments, respectively. Empty strings or non-alphabetical strings were removed from the resulting lists.

The second batch was comprised of PDF files (Vella, 2022). In the case of this batch, the corrected and uncorrected pairs did not exist on the same documents. Every exercise had four documents marked with A, B, C and D. The A files were the original texts with the mistakes, and the B files were the same but with the mistakes highlighted. The C files were the texts with the corrections expected to be applied by the students. The D files contained images and were thus not relevant to our process. We used *PDFMiner* to extract the text from the PDF files. We only needed incorrect-correct sentence pairs, so

either A-C text pairs or B-C text pairs were valid for our requirements.

In the case of both batches, some manual intervention was still required to get the files to a state where they could be used by the GEC tool. These were mainly circumstances when there were errors like using commas instead of full stops. The full stops, in particular, were needed to split the text into separate rows. Each sentence pair needed to be perfectly aligned as this was a basic expectation of the EncDec network for it to function correctly.

We decided to set aside the second batch (sourced from PDFs) as an evaluation set. Meanwhile, we parsed the files in the first batch to use the sequences as training data. This result consisted of approximately 4000 sentence pairs. The list was split into two files to train and validate the initial baseline models. Subsequent data sources contained grammatically correct data, so we would need to create copies with synthetic errors for them to be used by the model.

3.4.2.3 | MLRS Corpus

The MLRS made many different language resources available for Maltese, including tagged sets and lexicons (Gatt and Čéplö, 2013). We extracted the tagged sets of *Korpus Malti V3* for this study. We did not require any tagging information, so these files were parsed to reconstruct the original sentences. The sentences are assured to be grammatically correct, so the files were passed through an error synthesiser to create noise within the source text. The corpus data was used in the *Domain and Error Adaptation* phase of the experimentation.

3.4.2.4 | Common Voice

Common Voice[10] is an online platform that allows users to score a collection of voice clips (Ardila et al., 2020). The voice clips are people reading text that appears on the screen, and the users then mark whether or not they thought the voice clip matched the text on the screen. There are multiple datasets[11] available for different languages, and there is an available dataset for Maltese as well. We downloaded this dataset, mined the files for the sentences and produced a list of valid sentences. Like the MLRS corpus, we needed to create a duplicate with spelling errors synthetically. This dataset was also used in the *Domain and Error Adaptation* phase of the experimentation.

[10]https://commonvoice.mozilla.org/en (accessed: 2022-05-14)
[11]https://commonvoice.mozilla.org/en/datasets (accessed: 2022-05-14)

3.4.3 | Process

Marian's build includes some utilities that can be executed individually, including a vocabulary builder, a trainer, a decoder, a scorer and a converter. We used nearly all of the utilities for this study, save for the converter, which was only meant to convert Marian model files into a binary format. In later versions of Marian, such as the one used for this study, generated model files were stored in Npz files (output by the Python-based library, Numpy).

3.4.3.1 | Training

Marian is trained to translate by passing three pairs of data files (each would contain a file for the source and target languages). The first pair of files would be the training sets, and the toolkit would use these as inputs in the network's neurons. The second pair of files would contain the validation sets. After passing through the network, the training sets would be compared against the validation sets to determine the present accuracy of the model. The third pair of files would contain encoded vocabularies that contain aggregated tokens for the source and the target languages. The training and the validation sets are created using a random pick from the original dataset files.

Marian expects a set of parameters to fine-tune the network architecture when training. Marian supports a modern transformer architecture and a traditional Seq2Seq model. In the latter case, one could even define the type of cells being used, such as GRU or LSTM. Transformer models only allow the use of GRU cells. If required, developers can even influence the depth of the network and other intricate training parameters.

Even though Marian works with the same concept of *epochs* as with other machine learning engines, the toolkit validates the model after a predetermined number of *updates*. Updates are synonymous with *batch iterations*, i.e. the number of batches of data the model has processed. Marian always batches sentences rather than training on the entire dataset in one go. After a defined number of updates, the present model would be tested against the validation datasets according to validation metrics.

Marian automatically saves the best-performing model for each validation metric. In our study, Marian was configured to use many of the available validation metrics, including BLEU scores, perplexity, cross-entropy, and cross-entropy mean words. Therefore, the model was tested for each metric at each validation step and saved if they achieved a better score.

3.4.3.2 | Building Vocabularies

Originally, we attempted to construct the vocabularies ourselves in Byte-Pair Encoding (BPE) encoded files. BPE encoding works by splitting text into *subword units*. The idea behind subword units was conceived by Sennrich et al. (2016b) to deal with the limitation of having many unknown words by splitting up words into smaller parts. BPE achieves this sort of segmentation by encoding the most frequent common characters in a vocabulary.

Marian accepted the BPE-encoded files, but the model was not managing to improve its scores even after numerous iterations. We decided to introduce an alignment file as an additional parameter. Alignments are logical associations between the tokens of a sentence. For example, assume the sentence pair: 'The dog place with bawl' and 'The dog plays with the ball'. The token, 'place' and 'bawl', from the first phrase should ideally align with 'plays' and 'ball' in the second phrase, even though the spelling and sentence lengths are different.

The alignment file is used to bias the attention mechanism in a transformer architecture towards specific word alignments over others (Chen et al., 2016). It was discovered that sequence models do not consistently achieve the ideal alignments, so guiding the attention mechanism could prove beneficial to the model both in terms of accuracy and efficiency (Bahdanau et al., 2014). Typically, an alignment would have required a trained model to achieve correctly, but this only held for when one was translating between two completely different languages. The probability of having a near-perfectly sequential alignment would be very low. In the case of our GEC tool, we did not have this limitation.

We had the option to rely on calculating the alignment using unsupervised methods such as contextual embeddings. Contextual embeddings recommend alignment based on sequence-level semantics and similarity between two strings. Using this method, we used SimAlign (Jalili Sabet et al., 2020) to generate an alignment file between our source and target languages and managed to generate a very satisfactory alignment file. Experimentation with the new alignment file only improved the results but only marginally.

After consulting the documentation, we discovered that Marian could work more effectively with vocabularies built using *SentencePiece*. SentencePiece is an unsupervised text tokeniser meant for neural networks, and it is most applicable when the vocabulary size can be determined prior to training. SentencePiece uses BPE-encoding and develops an alignment pattern after training on the raw input sentences themselves (Kudo and Richardson, 2018).

Rather than download the official distribution, we opted to use the *marian-vocab* utility, which maintained its own version of the library. There are only a few differences, such as a slightly more powerful performance and Marian's version appends some unique tokens to the vocabulary that Marian's models require. Otherwise, we would have needed to append these tokens ourselves to get the network to train. Using Marian's vocab utility, we generated our SentencePiece vocabularies based on the source and target datasets. We then appended these vocabularies to the model parameters. This particular configuration registered significant improvements to the scores during training.

3.4.3.3 | Translation

The decoder accepted a trained model as a parameter so it could then translate a text file. Marian allowed the decoder to accommodate changes similarly to how the model could be fine-tuned during training, including changes to the depth of the neural network. It could also output additional helpful information, such as the alignment of tokens determined by the attention mechanism. The attention weights were represented using decimals (between 0 and 1), representing a dependency between a token and the other tokens. The decoder could be configured to accept different models at once and produce simultaneous translations per sentence, which would prove helpful when comparing against multiple models. Furthermore, the decoder could be fine-tuned to exercise bias, favouring the results of one or more models by raising their weighting ratio.

3.5 | Experimentation Strategy

Due to time constraints and the imposed limits of use when interacting with Google Colab, we had to be pragmatic and develop a strategy that prioritised the first two objectives of our aim. We decided to apply three variant encoder-decoder models to achieve the first objective, as detailed in Subsection 1.3.1. Referring back to the later sections of the literature analysis in Chapter 2, it was clear that EncDec architectures were not only the most widely used architectures but also the most successful and the best performing. In fact, nearly all the participating solutions in the BEA-2019 shared task (as detailed in Section 2.9) were based on EncDec architectures and produced state-of-the-art results. The dilemma was, therefore, whether to adopt a traditional Seq2Seq model or opt for a variant of the more recent transformer model (a Seq2Seq model with the attention mechanism).

The literature suggested that the transformer should outperform the Seq2Seq model. We decided to test out this hypothesis by developing a selection of baseline models trained on the same data and for the same number of epochs so that we could draw comparisons between their performances. For this experimentation phase, we decided to train with the first dataset at our disposal, which was the collection of corrected Word documents made available by Vella (2015). These were further elaborated upon in Subsection 3.4.2.2. Marian NMT supports several trainable model types, all of which are variants of EncDec architectures. For our purposes, we decided to implement the following three base models:

- Model-Type AMUN (Nematus): This is a transformer implementation that is functionally equivalent to Nematus Sennrich et al. (2017).
- Model-Type Transformer: This is a transformer implementation that is functionally equivalent to the implementation of Vaswani et al. (2017).
- Model-Type S2S (Seq2Seq): This is a Seq2Seq model that is functionally equivalent to DL4MT[12]. DL4MT was mentioned in Sennrich et al. (2017) as the base system upon which Nematus was built, before the implementation of the GRU-based attention mechanism.

After establishing a baseline model, we focused on achieving the objective detailed in Subsection 1.3.2 by applying the adaptations detailed in Junczys-Dowmunt et al. (2018b). After each adaptation, the model was retrained and re-scored. The model scores were then compared to the previous incarnation to check if the quality of the model had improved.

Not every adaption in the original study could be replicated faithfully. For example, the original study used different monolingual pretrained language models to influence the decoder parameters of the translation model. We did not have such models in our possession and could not replicate those experiments. Furthermore, the original study described the use of Edit-Weighted MLE to influence the weighting of the network. The exact method of incorporating this into the model was unclear and could not be replicated. We were left with Source Word Corruption, Domain and Error Adaptation, Tied Embeddings and Pretrained Embeddings. During our research, we also came across mentions of how very large vocabulary sizes become counterproductive and start diminishing the quality of the model. We decided to add an experiment to check this phenomenon as well.

[12]https://github.com/nyu-dl/dl4mt-tutorial

3.5.1 | Scoring Strategy

The default scoring used by Marian is the BLEU score. The BLEU metric scores the similarity between two sentences and cites that value as a percentage or a decimal between 0 and 1 (Papineni et al., 2002). BLEU seems to correlate quite strongly with human translators.

On a technical level, BLEU works using two metrics: *Overlap* and *Sentence Brevity Penalty* (Papineni et al., 2002). Overlap is a modified precision metric based on N-grams. A value is considered to be precise if it is observed in the target phrase. However, that would mean that the phrase 'The the the the the' would achieve a perfect score when evaluated against 'The dog likes the ball' because the word 'the' exists in the target sentence, ergo, every word in the first phrase exists in the second phrase. Such a precision score would be misleading because, in truth, the first phrase is an awful translation of the second phrase.

Overlap rectifies this issue by only giving credit to a token as many times as it appears in a phrase (the earlier example would only score 20%). Furthermore, BLEU does not only check at a uni-gram level but does the same exercise on bi-grams and tri-grams. At least one match of 4 tokens in a phrase is required to achieve any BLEU score higher than zero.

The brevity penalty lowers the achieved score if there are differences between the sizes of the compared phrases. That way, the phrases "The dog likes the ball" and "The dog likes the ball and the bone" would not achieve a perfect score even though the former phrase entirely exists in the latter.

BLEU performs best when scoring a corpus rather than individual sentences and requires normalised and tokenised input to produce the best results. BLEU also does not distinguish on a semantic level. For example, the words 'dormouse' and 'doorway' would be incorrect translations of 'doorbell' even though 'doorway' arguably arrived closer to the word's actual meaning. In the context of GEC, this limitation is not very crucial since the only interpretation of a correct word is if it is spelt exactly the way it should be.

3.5.2 | Hyper-Parameters

In Marian, training could occur either on an epoch or iteration basis. An epoch is when all the observations in the training sample space are processed, forward and backwards, in the neural network. An iteration is when the observations within a *batch* are processed. Batching has been explained briefly in Subsection 3.4.3.1 in the previous chapter.

Marian was configured to initialise a scoring phase for every 500 batch updates for training. During scoring, checkpoints were saved to a mounted Google Drive storage space. By default, Marian only keeps the best-scored model files during run-time, so checkpoints would not be overwritten unless the last training batch yielded better scores than the previous one. Furthermore, different model files are stored depending on the best scores, and Marian writes a separate model file for each scoring method. So if, for example, the last checkpoint registered a weaker BLEU score and a better perplexity score, Marian would only overwrite the model file relating to perplexity.

We initially tested the maximum vocabulary sizes possible for our datasets. The vocabulary sizes were initialised to approximately 14,500 entries for the source language and 13,800 entries for the target language, which were the appropriate sizes for the first batch of files extracted from the Qari tal-Provi dataset as explained in Subsection 3.4.2.2. The vocabulary sizes would change as we implemented more adaptations, which in most cases required additional datasets from our pool of resources. The vocabularies are necessary for the models for both the encoder and decoder as they dictate the size of the input and output parameters of the neural network.

The rest of the hyper-parameters were initialised as follows. Note that specific arbitrary configurations related to debugging and I/O, such as logging options and save paths, were omitted from the following list.

- **train-sets** - This option points Marian to the training data. Before the commencement of training, we split our dataset into separate training and testing sets. We included the source and target sentence pairs in separate files for each set. We passed the paths of the training sets (both the source and target files) into this parameter.
- **valid-sets** - Conversely, this option is used for the validation data. We passed the paths of the validation sets (both the source and target files) into this parameter.
- **vocabs** - Marian expects the paths for any vocabulary files used. We passed the paths of the vocabularies generated using *SentencePiece* into this parameter. There were two vocabularies, one for the source language and one for the target language. Even though we were using Maltese throughout, our source language was littered with incorrectly spelt words, many of which had missing Maltese fonts. These instances had to be present in the source language vocabulary so the encoder could still translate them to vectors. At the same time, they should not be present in the target language vocabulary because the decoder should not trans-

late any vector representation into an incorrectly spelt word. Incorrect Maltese and correct Maltese should be treated as separate languages.

- **workspace** - This option determines the amount of RAM that the toolkit uses when training. We set this value to 10Gb, which would occupy roughly 80% of the available memory on the machine.
- **cpu-threads** - Marian was set to spawn 8 individual CPU threads while training. This option was only used when training on a machine that did not have a GPU
- **num-devices** - This option determines the number of GPUs that could be used during training. The value for this option was always set to 1 since we never had the possibility of enjoying a multi-GPU setup during training. Marian will revert to the previous parameter if it does not have a GPU.
- **mini-batch** - This is the minimum batch size Marian is allowed to designate during training. The batch size is communicated in terms of the number of sequences. This value was set to 4 as the official Marian documentation and online forums recommended.
- **maxi-batch** - Conversely, this option represents the highest possible batch size. This value was set to 100 as the official Marian documentation and online forums recommended.
- **max-length** - By default, Marian caps the number of tokens it can process at a given time since Marian's supported architectures separate sentences into shorter phrases during training. The value of this option determines the maximum length of these phrase sequences. We set this value to 100, which is twice the default. While higher max lengths impact performance, our initial training datasets were relatively small, and we preferred to aim for better quality in the model.
- **layer-normalisation** - This option allows Marian to perform a data normalisation process on the values of the activation layers within the neural network. This option can only be turned on or off and accepts no further optimisations. We activated it for our experiments.
- **dim-emb** - This option controls the size of the word embeddings dimension. We kept the default size of 512 for most of our experiments. The value was only bumped to 768 when we commenced with pretraining from pretrained BERT models.

- **enc-cell** - This configures the type of encoder cell. All transformer implementations supported by Marian use GRU cells. Seq2Seq models can support a different cell type such as LSTM.
- **dec-cell** - This configures the type of decoder cell. The same limitations apply depending on the transformer or Seq2Seq setups.
- **dropout-rnn** - This option controls the rate of dropout in the hidden RNN layers. Dropout is an important consideration that allows the network not to become overly dependent on certain neurons. The value should not be too high as that would render the network incapable of making any prediction. We set this value to 0.2, corresponding to a 20% probability rate, as recommended by the literature Junczys-Dowmunt et al. (2018b).
- **dropout-src** - This is the same concept as the previous option but applicable to input source words. We set this value to 0.1, corresponding to a 10% probability rate, as recommended by the literature Junczys-Dowmunt et al. (2018b).
- **dropout-trg** - This is the same concept as the previous option but applies to output target words. We set this value to 0.1, corresponding to a 10% probability rate, as recommended by the literature Junczys-Dowmunt et al. (2018b).
- **tied-embeddings** - This option controls weight typing between the embedding layers and the SoftMax layers. It can be done for either the destination layer (this option) or the source layer (*tied-embeddings-src*), or both (*tied-embeddings-all*). We initialised the model with tied embeddings for the destination layer only, which was a basic necessity to get the model working. Otherwise, the model architectures would not be able to conduct their output in a form compatible with our Maltese language vocabularies. We would apply tied embeddings across both the destination and the source layers in future experiments.
- **seed** - This option introduces a seed that initialises the initial weights of the network. The initial weights are determined using random number generators, but the seed would guarantee the same initialisation state for the network. In order to evaluate our baseline models, we needed to remove as many elements of randomness as possible. We, therefore, initialised this value to 1111 (any other value would have also sufficed, but we stuck to the most commonly used one in the examples used in the official documentation).
- **overwrite** - Marian was set to overwrite the checkpoint file rather than write a new one each time.

- **keep-best** - Marian was set to keep the best checkpoint file rather than ever overwrite with a weaker resultant model file.
- **no-restore-corpus** - Marian was set not to restore corpus positions in the event of resuming training on a different day. Splitting the datasets was done using a random seed, so this measure ensured that the model did not overfit a particular part of the corpus.
- **optimizer** - Marian supports three optimiser types: Adam, Stochastic Gradient Descent (SGD) and Adaptive Gradient Algorithm (Adagrad). All of our models were optimised using Adam since it was the most commonly adopted optimiser in the literature Junczys-Dowmunt et al. (2018b).
- **optimiser-params** - Since the literature did not indicate specific parameters, we kept the default parameters since they were the same parameters used in the publicly available code branch related to the study of Grundkiewicz and Junczys-Dowmunt (2018). The exponential decay rate for the first moment estimates, Beta 1, was set to 0.9. The exponential decay rate for the second-moment estimates, Beta 2, was set to 0.98. It is recommended that Beta 2 is set to a value close to 1 in NLP tasks. The last parameter, Epsilon, used to avoid divisions by zero, was set to $1*10^{8}$.
- **learn-rate** - The learning rate used when adjusting the gradient. This value was set to 0.0001.
- **beam-size** - This controls the size of the beam search window as described in Section 2.5. We set this to 24 to match the literature Junczys-Dowmunt et al. (2018b).
- **valid-reset-stalled** - If a metric stalls and stops showing improvement, the metric is recalculated.
- **valid-freq** - The frequency at which Marian would perform validation scoring. We set this to the rate of every 500 updates. Realistically, this could be higher or lower depending on how deep one would want to analyse the rate of improvement of the model.
- **valid-metrics** - Marian supports a number of validation metrics. We used BLEU as the primary metric for scoring the model. We also used cross entropy, mean words and perplexity as supporting metrics.

3.6 | Summary

This chapter elaborated thoroughly on the implementation strategy adopted for this study. We decided to focus on developing the EncDec models. This decision was heavily influenced by the previous literature review, which clearly outlined the advantages of Seq2Seq and transformer models over earlier machine translation methods.

We decided to use the Marian NMT toolkit to create our models. The toolkit was one of the leading publicly available solutions in Neural Machine Translation. Marian NMT not only supported renowned model architectures such as the ones of Vaswani et al. (2017) and Nematus (Sennrich et al., 2017) but was also the basis of several studies that achieved state-of-the-art results (Grundkiewicz and Junczys-Dowmunt, 2018; Grundkiewicz et al., 2019; Junczys-Dowmunt et al., 2018b). Marian NMT was a vital part of the leading submission in the BEA-2019 shared task.

We decided to rely on Google Colab for our hardware infrastructure. The platform allowed us to leverage GPUs, TPUs and CPUs for training. Google Colab is written in Python, the language in which Marian NMT is written (even though it is built in C++).

We prepared some different datasets for training and evaluation. These include document pairs that Maltese diploma students corrected, extracts from an online platform named *Common Voice* and the MLRS corpus. We sometimes even generated synthetic data using a custom-built error synthesiser. We did this in support of several adaptations we applied to the models, such as error adaptation. Other adaptations involved purposely corrupting the source files, weight tying in the embedding layers, and pretraining the network weights. For pretraining, we acquired two language models based on BERT that had been trained for Maltese. There was a monolingual and multilingual model called BERTu and mBERTu, respectively.

For training, sentence pairs were passed to the model architectures in source and target language files. The models were also supported with vocabulary definition files for either language. These vocabularies were generated using a tool named *SentencePiece*. The translation was done using Marian, which had its native decoder function.

4

Results

This chapter describes the different experiments and the parameters we applied during the experimentation. As was explained in the previous chapter, we used the Marian NMT toolkit to create a collection of encoder-decoder models. This chapter will only focus on observations and conclusions that could be drawn from the training process itself. Discussions relating to the solution's real-world evaluation and performance will be dissected in the subsequent chapter, Chapter 5, which focuses on evaluating the model on unseen authentic datasets.

After trials and critical analysis, we arrived at the final model. We based the design on the work of Grundkiewicz and Junczys-Dowmunt (2018) who leveraged Marian for low-resource GEC during the BEA-2019 shared task. We then based our experimentation strategy on the work of Junczys-Dowmunt et al. (2018b) who had applied sequential adaptations on a baseline model until arriving at the most potent setup that could perform in low-resource scenarios. In some cases, it was not possible to perfectly replicate each experiment. In other cases, the experiments did not yield the results that we were inclined to expect from the original study's findings. These will be discussed further in detail.

This chapter is divided as follows. Section 4.1 illustrates the different scores gathered for the different modelling experiments. Visualisations were included to assist with the interpretation of these results. Section 4.2 contains a summary of the chapter's findings.

4.1 | Model Scores

This section discusses the results obtained during experimentation. For each subsection, we obtained scores for BLEU, cross-entropy, cross-entropy mean words and perplexity.

The subsequent subsections will only discuss the BLEU score results. The rest of the results can be found in Appendix 6.4.

4.1.1 | Baseline Architectures

Figures 4.1 and 4.2 depict the BLEU scores obtained during training per 500 batch iteration updates across 20,000 iteration updates. As seen in Figure 4.1, there were instances when the models would diminish their quality after a set of updates. Figure 4.2 depicts the climbing BLEU scores when considering only the best scores achieved at any given batch.

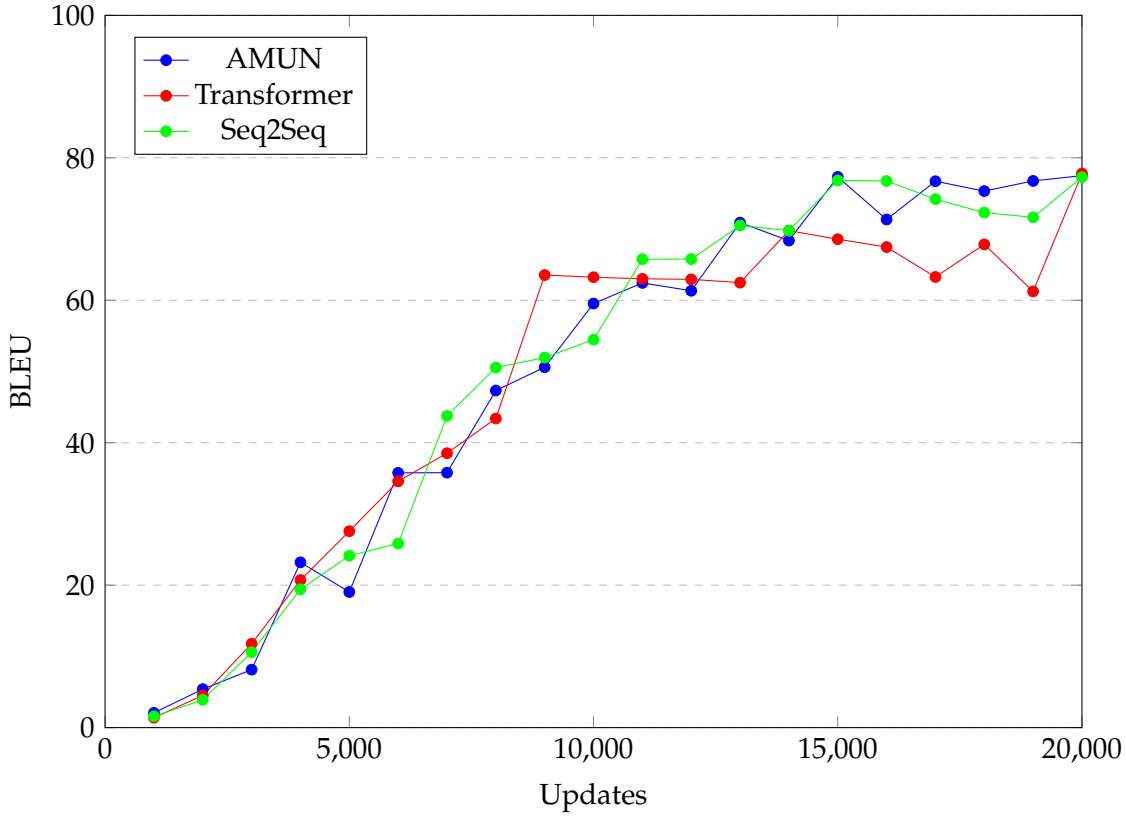

Figure 4.1: Baseline Architectures - Running Scores

The quality of all three models happened to be relatively consistent. It is possible that when dealing with such low volumes (at this stage, we only had 4000 sentence pairs), it was difficult for one implementation to overtake another significantly. The progression of the BLEU scores paints a different picture of the models' behaviour. In the case of the Vaswani transformer, it reached a plateau at the 9000th iteration and at the 14,000th iteration. The most consistent improvements came from the Nematus transformer (AMUN)

and the Seq2Seq model. The top scores were 78.11%, 77.79% and 78.36% for the Nematus transformer (AMUN), the Vaswani transformer and the Seq2Seq, respectively.

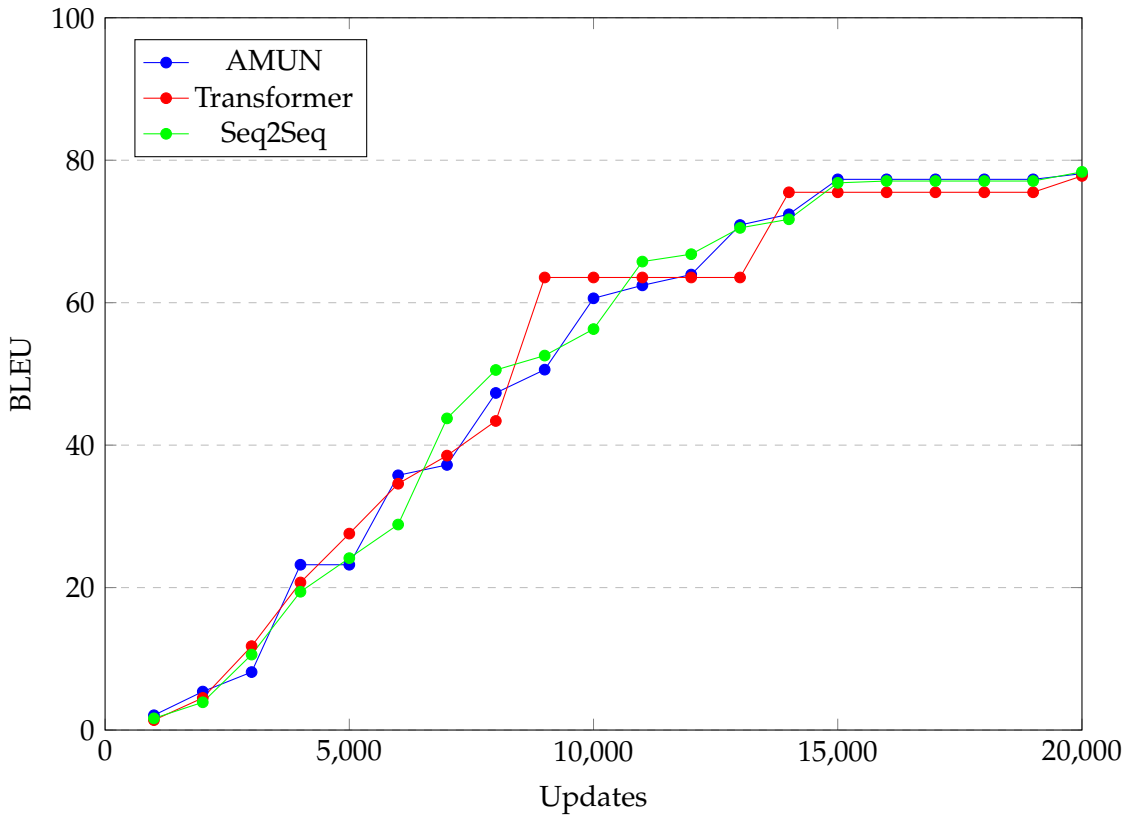

Figure 4.2: Baseline Architectures - Best Scores

We needed to select one of these models to apply adaptations because we did not have enough time to investigate the impacts of the adaptations on each implementation individually. While Seq2Seq outperformed AMUN, it also had the highest computational costs. On a typical Google Colab session, we could spawn eight individual processor threads for AMUN training, but the same measure would cause a memory crash if attempted for Seq2Seq because the machine would run out of memory. In fact, the machine could only handle half the threads at most when using a Seq2Seq architecture. An advantage of just 0.25% did not merit this downgrade in efficiency, so we continued building subsequent models based on the AMUN architecture.

4.1.2 | Source Word Corruption

We used our custom-built error synthesis tool to drop some source words from the training files. The original study did not highlight any particular routine for this kind of corruption, so we opted to stick to the rate that was most commonly used in other NMT projects. We thus configured the synthesis tool to remove every tenth word from the source files. As one can observe in Figure 4.3, the result was a steep drop of nearly 10% in the BLEU score, which totalled up to 69.26%. We suspected that the reason behind this was the lack of training data. While the technique was initially implemented for a low-resource setting, the original authors still worked with at least 20,000 sentence pairs compared to our mere 4000. We deduced that the drop in BLEU score occurred due to removing too much information from the source files. We reverted the model to its previous state before proceeding with the following adaptation.

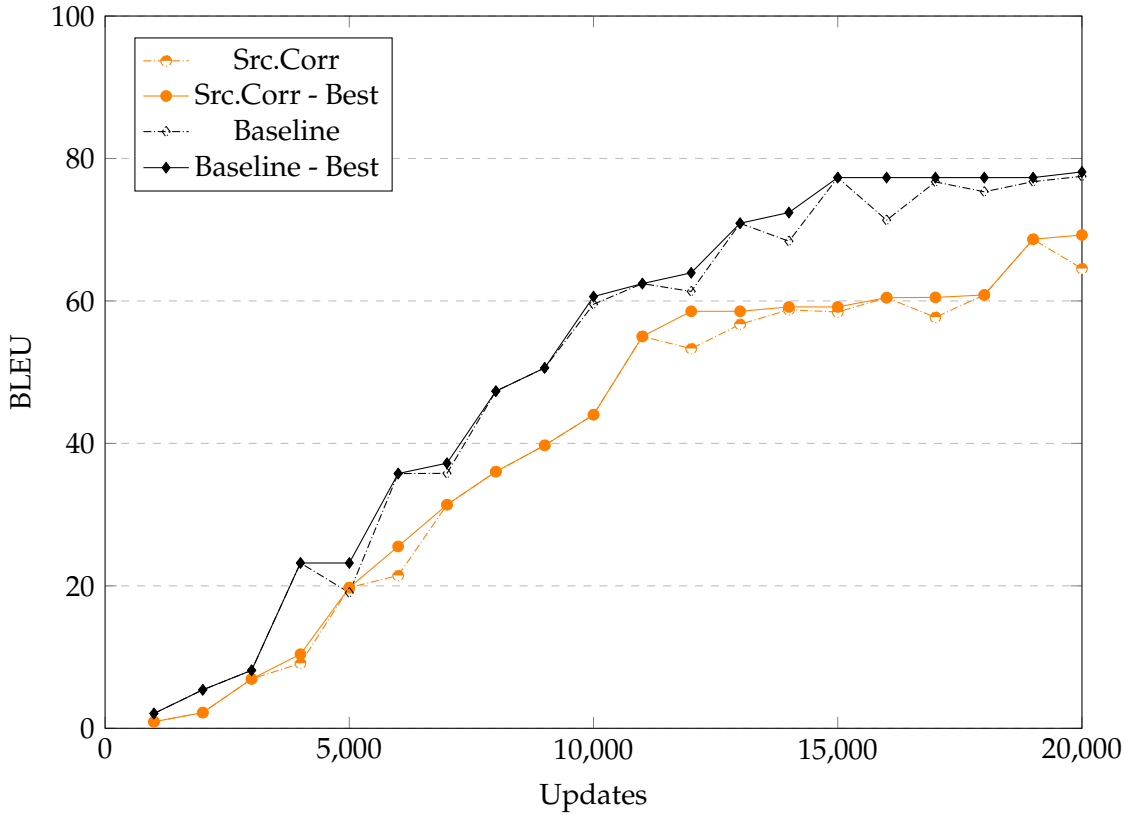

Figure 4.3: Source Word Corruption

4.1.3 | Domain & Error Adaptation

In the original study, domain adaptation and error adaptation were applied separately. However, this was only possible because their sources had already contained sentence pairs with factual errors. The only source we had at our disposal with real-world errors was the corrected document pairs from Qari tal-Provi. All our remaining sources (MLRS and Common Voice) did not have errors. Therefore, our only option was to increase the size of our training data by taking the original files and introducing noise into them. At the same time, these additional sources introduced new domains into our training space. Therefore, domain adaptation and error adaptation were applied simultaneously to our models.

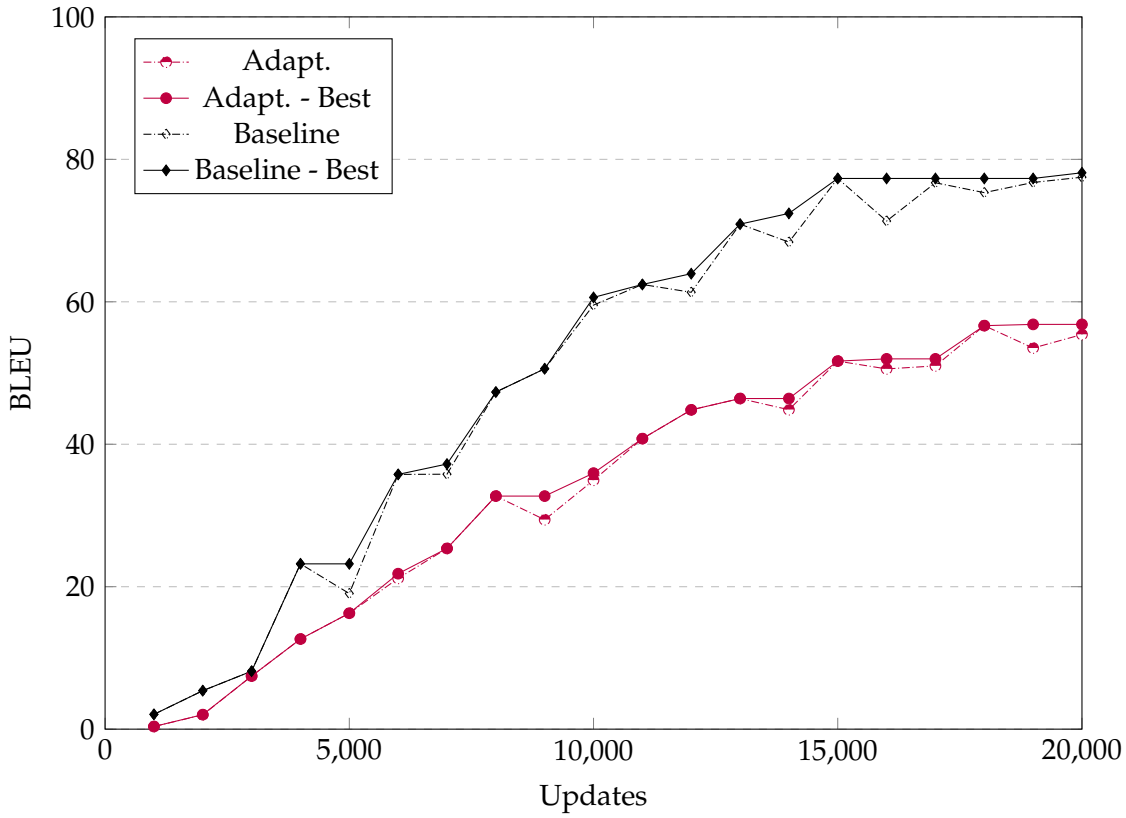

Figure 4.4: Domain & Error Adaptation on Baseline

Domain adaptation refers to the inclusion of data relating to new domains. Domain adaptation allows the GEC solution to become more generalised rather than specialising in one area. Care would be taken to ensure that the dataset sizes were similar and not too far off from each other. We were fortunate that the additional sources had row counts close to the first dataset. The amalgam of the Qari tal-Provi (Batch 1), Common

Table 4.1: Error Seeds

Error Strategy	Sentence Seed	Token Seed	Character Seed
Common Article	0.5	0.2	-
No Silent Letters	0.4	0.4	-
No Fonts	1	1	-
Key Proximity	0.2	0.1	0.05
Insertion	0.2	0.1	0.05
Duplicate	0.2	0.1	0.05
Substitution	0.2	0.1	0.05
Omission	0.2	0.1	0.05

Voice and the MLRS datasets yielded a final large dataset of approximately 13,000 observations.

Error adaptation is the practice of tweaking the error rates in one's separate data source such that all the datasets have approximately the same number of errors. We did not have any errors in our newly introduced datasets, as stated beforehand. We, therefore, leveraged the error synthesis tool a second time to create noise in our new files. First, we analysed our first dataset (the corrected document pairs) to determine the most common errors in the files. There were various spelling mistakes. However, the most common error by far was the absence of Maltese fonts. The remaining errors included the article, keyboard proximity, and capitalisation errors.

We applied error synthesis on our Common Voice, and MLRS detests. These datasets were described in Subsection 3.4.2.3 and Subsection 3.4.2.4 respectively. The error synthesis tool was described in Subsection 3.4.2.1. For error synthesis, and all the text was cleaned of Maltese fonts. A series of random seeds were implemented to create additional errors. Table 4.1 shows the full list of seeds.

This adaptation on top of the model achieved a BLEU score of 56.82% as referenced in Figure 4.4. This value was smaller than the established baseline. We had originally expected a larger BLEU score from this model, but it is important to consider that the dataset was much larger. The drop in the overall score was because of the added data volume. A more likely explanation is that the errors present in the latter two datasets were not sourced from authentic, real-world sources but were artificially introduced.

4.1.4 | Larger Vocabularies

We decided to test out the claim that large vocabularies could impact the quality of the model by boosting the vocabulary sizes for both the source and target languages. We increased the size from 14,500 to 30,700 entries for the source language, and we increased the size from 13,800 to 22,000 entries for the target language. The increase in vocabulary size reduced the overall BLEU score from 61.68% to 52.55%. Another interesting observation was that the training did not stall using larger vocabularies. As evidenced by Figure 4.5, almost every batch iteration registered an improvement. At the very least, using larger vocabularies might guarantee a more stable rate of improvement in the training process. Nevertheless, our original hypothesis was verified since the previous benchmark registered a stronger BLEU score.

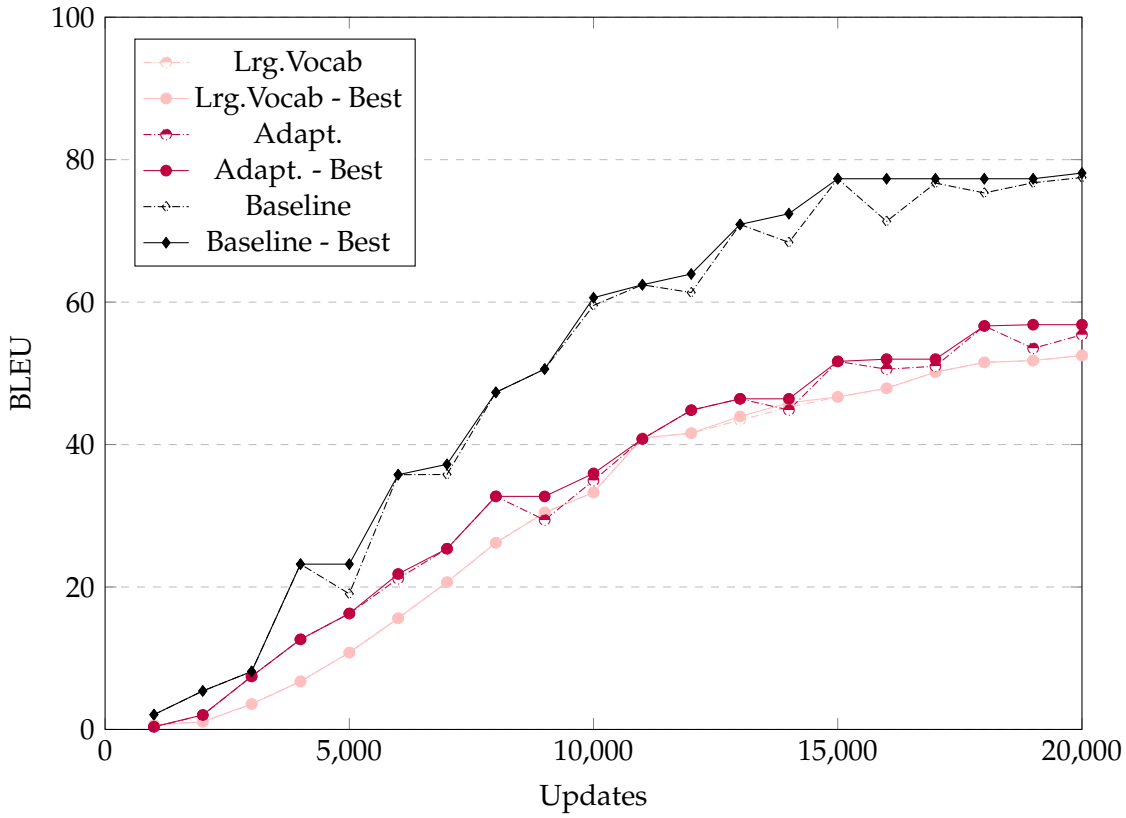

Figure 4.5: Large Vocabularies on Domain & Error Adapted

4.1.5 | Tied Embeddings

In the previous adaptation, we suspected the BLEU score had diminished due to the increase in the data, which also increased the size of the training set and the validation set. We decided to run the experiment on the domain-adapted model and the original benchmark model to demonstrate the benefits of tied embeddings.

We needed to change the vocabulary sizes again to tie the weights between the Soft-Max layers and the source and destination layers for this adaptation. The sizes of both source and destination vocabularies needed to be the same. We thus limited the size of either vocabulary to 22,000 entries for the adapted model and 13,000 entries for the baseline model.

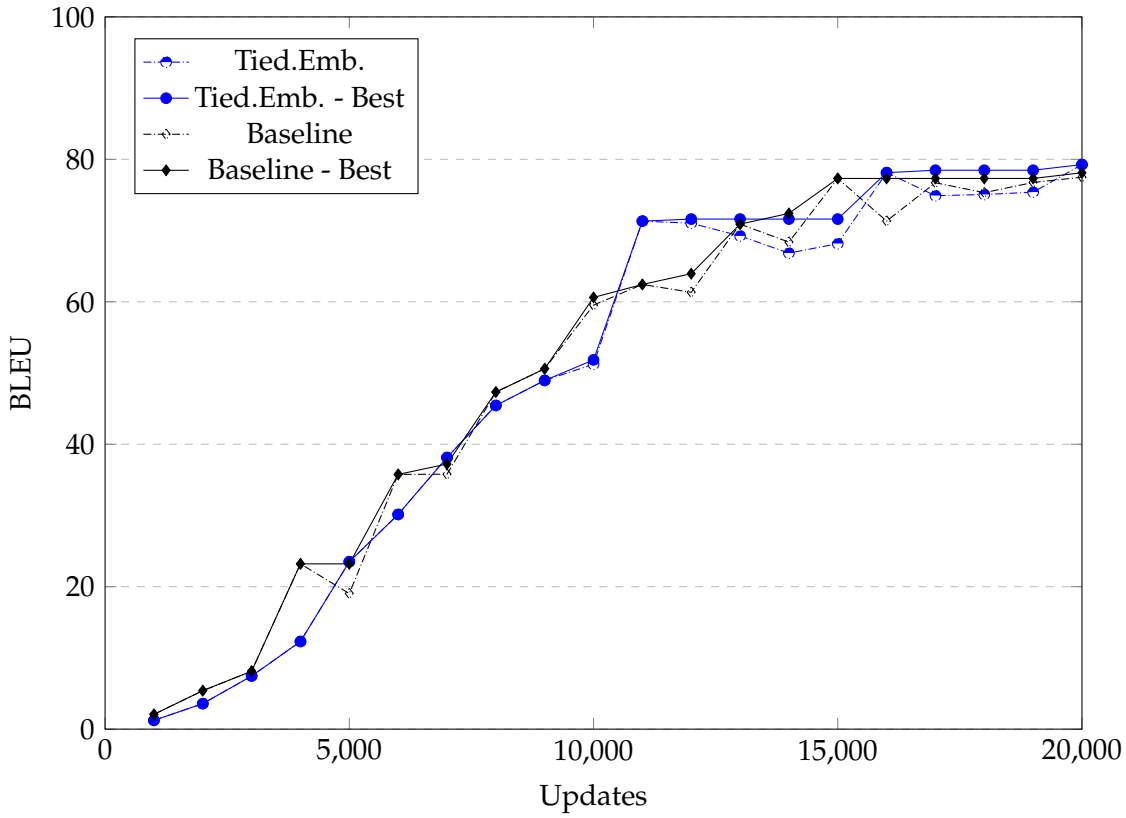

Figure 4.6: Tied Embeddings on Baseline

As shown in Figures 4.6 and 4.7, tied embeddings registered an improvement in the model's output for both the baseline and the domain/error adapted model. The model achieved a BLEU score of 79.26% when applied to the baseline model and when applied to the domain and error-adapted model. Due to the blanket improvement, we adopted tied embeddings as an additional layer of complexity over our existing models before

applying the last adaptation discussed in Subsection 4.1.6 i.e. pretrained embeddings.

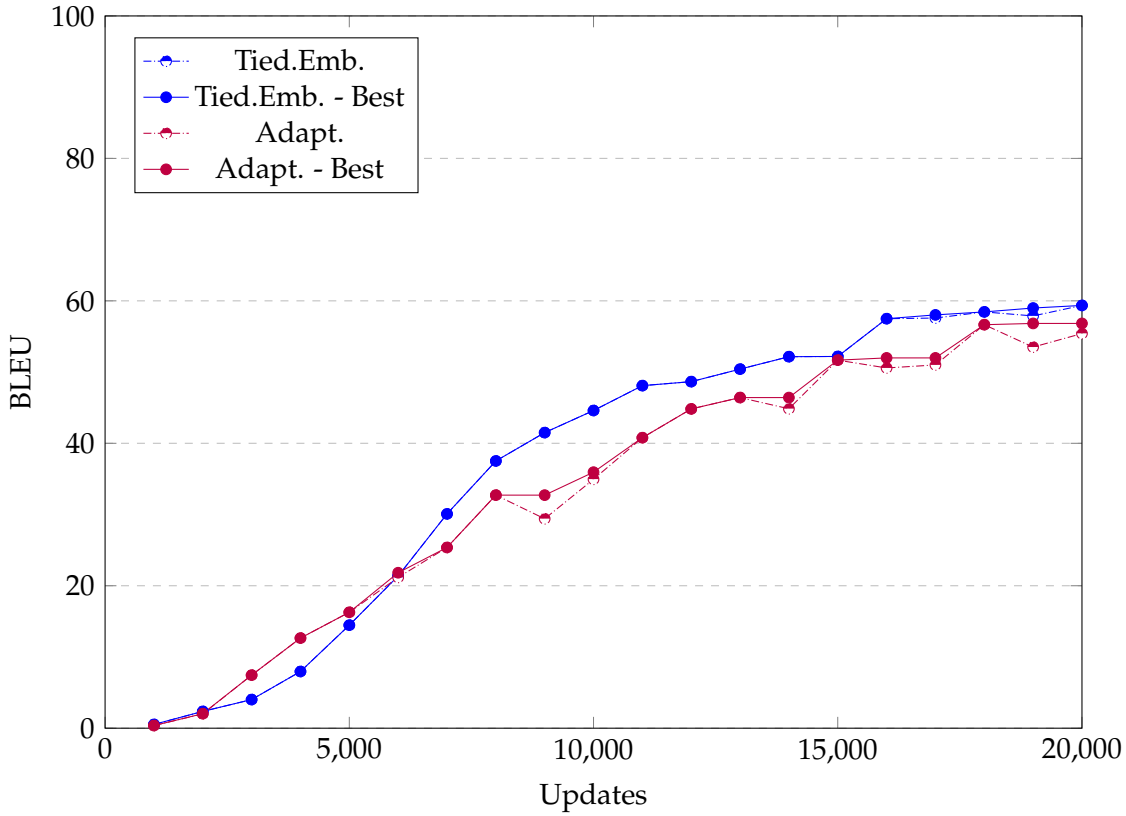

Figure 4.7: Tied Embeddings on Domain & Error Adapted

4.1.6 | Pretrained Embeddings

We attempted a transfer-learning-oriented adaptation for our last experiment that started the training process from a pretrained language model. For this phase of experimentation, we relied on the provided BERT models that we managed to acquire, as explained in Subsection 3.2.1.2. The two models, *BERTu* and *mBERTu* were both BERT language models that had been trained in the Maltese language. The first model was trained on a monolingual corpus, while the second model was further trained on a multilingual BERT model named *mBERT*.

In order to pretrain our model, we needed to switch back to a Seq2Seq architecture. We had to switch because of inherent incompatibilities between the BERT architecture and the transformer architecture implementations that existed on the Nematus (AMUN) transformer and the Vaswani transformer on Marian. However, considering how close the quality of the established baseline models was, this was not a problematic update for us to introduce.

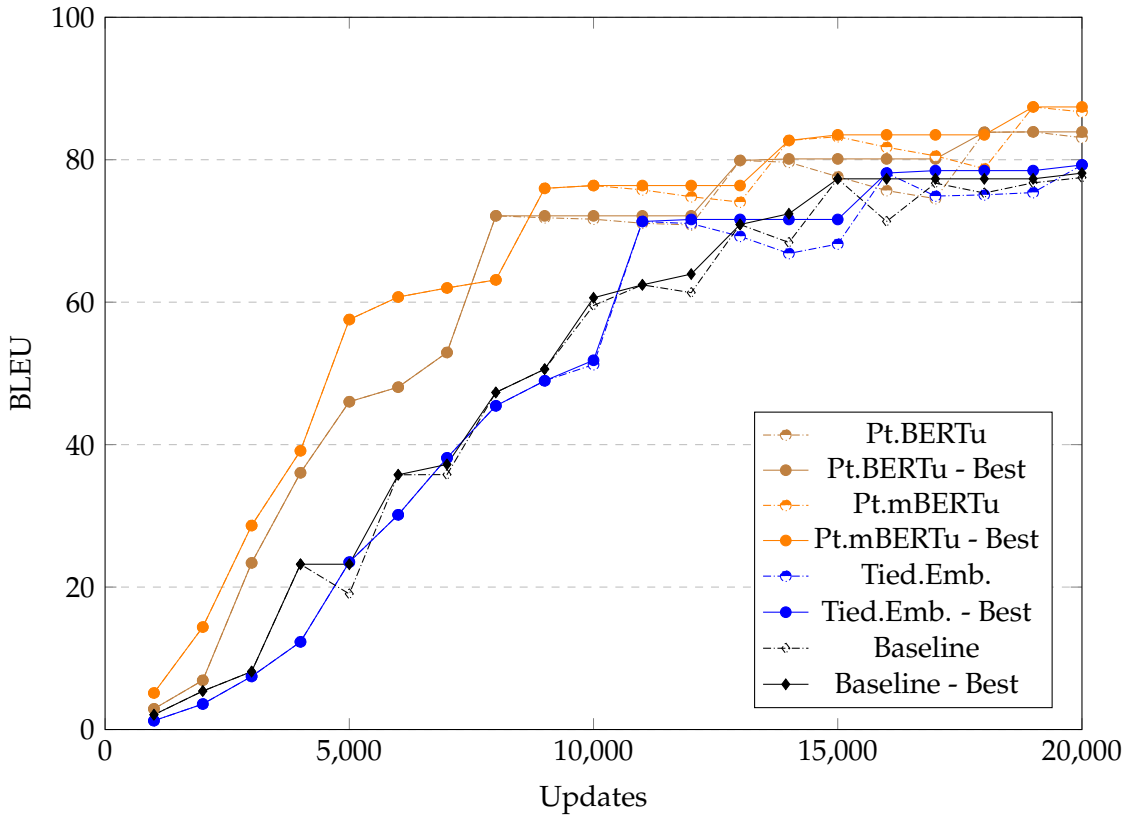

Figure 4.8: Pretrained Embeddings on Baseline

Both exercises were executed with the tied embeddings adaptation, which had al-

ready demonstrated an improvement. Like before, we conducted experiments on both the baseline and adapted models.

When trained on the monolingual model (BERTu), we obtained a BLEU score of 83.90% on the baseline model and 69.15% on the adapted model. When trained on the multilingual model (mBERTu), we obtained a BLEU score of 87.41% on the baseline model and 60.81% on the adapted model. As evidenced in Figures 4.8 and 4.9, both pretrained models surpassed the results obtained by the tied embeddings on their own. In the case of the model adapted with synthesised data, the monolingual was the strongest as evidenced by Figure 4.9. These findings were consistent with the literature, which maintained that pretraining a model from a monolingual model was more likely to produce higher-quality translation models than multilingual ones. In the case of the baseline models, we observed the opposite. Referring to Figure 4.8, the multilingual model surpassed the monolingual model.

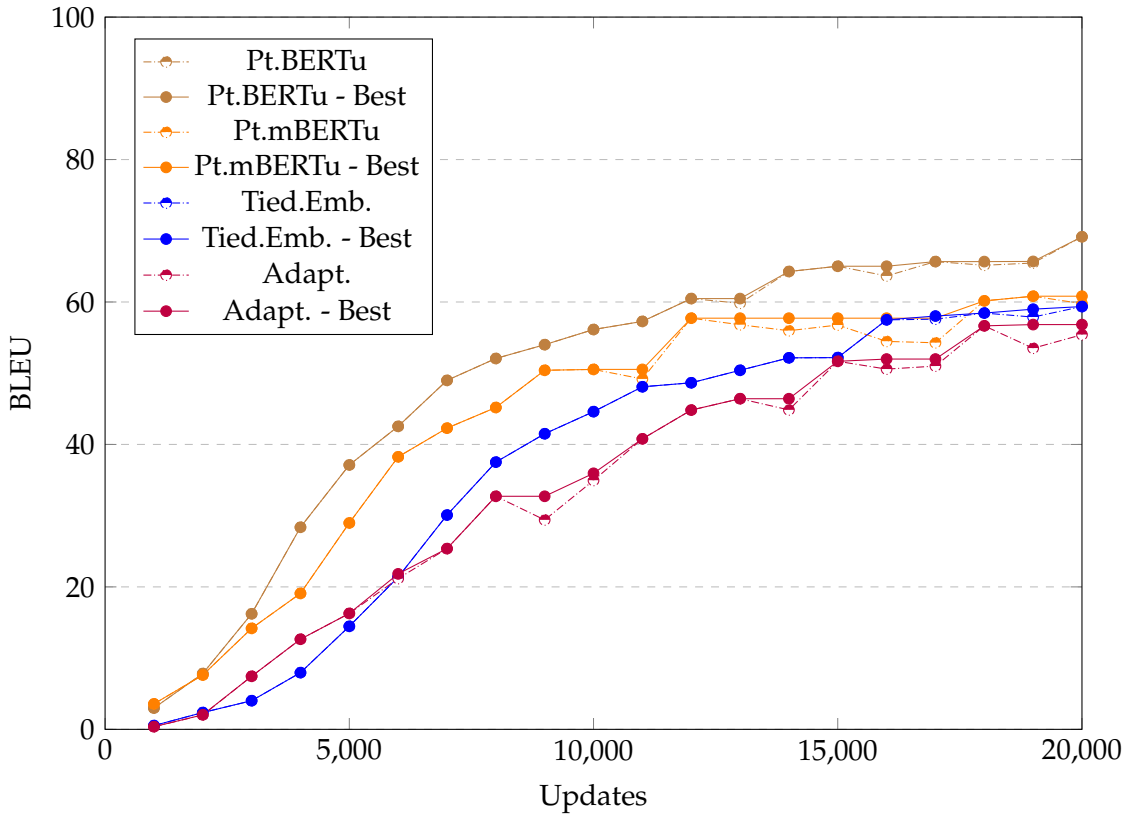

Figure 4.9: Pretrained Embeddings on Domain & Error Adapted

Table 4.2: Model Scores

Model Adaptation	BLEU
Baseline - S2S (Seq2Seq - GRU)	78.36
Baseline - Transformer (Vaswani)	77.79
Baseline - AMUN (Nematus)	78.11
Baseline + Src. Word Corruption	69.26
Baseline + Tied Embeddings	79.26
Baseline + Pretrained - BERTu	83.90
Baseline + Pretrained - mBERTu	**87.41**
Domain & Error Adapted	56.82
D & E. Adapted + Large Vocabularies	52.55
D & E. Adapted + Tied Embeddings	59.35
D & E. Adapted + Pretrained - BERTu	**69.15**
D & E. Adapted + Pretrained - mBERTu	60.81

4.2 | Summary

We concluded from the experiments that with small volumes of data, both Seq2Seq models and transformer models performed very well. A larger volume of data might have been needed to outline intricate details between these architectures truly. The influx of synthesised data diminished the model's accuracy, though this could have been a natural reaction to catering to new data domains. Source word corruption also negatively impacted the overall scores. It is possible that when the volumes of data are so small, applying source word corruption could remove too much information from the data sources and invariably impact the precision of the trained model. Vocabularies could also hamper the quality of the model if their size becomes too large. On the flip side, tied embeddings improved the quality of the correction model. Pretraining from language models trained on the same language also improved scores, even if the language models were not monolingual. The best-performing models included the tied embeddings adaptation and the pretraining model adaptation. The baseline models benefited most from the multilingual language model, but the domain-adapted model benefited from the monolingual language model. Table 4.2 depicts the results for each adaptation.

5

Evaluation

In this previous chapter (Chapter 4), we described how various Encoder-Decoder models were calibrated and trained on sets of data - some of them were authentic, while others contained synthetically generated errors. This chapter explores the process conducted to evaluate the system. We decided to use data to allow the solution to be evaluated in a real-world context. This approach gave us the best guarantee of a feasible prototype for a GEC model and allowed us to achieve our last objective defined in Subsection 1.3.3.

The chapter is structured as follows. Section 5.1 details the adopted strategy for evaluation. Section 5.2 details our analysis of the obtained results. Section 5.3 discusses some of the issues we noticed in the trained models. Section 5.4 focuses on our attempt to build a final model based on all the championed adaptations and trained on data that was more closely related to the domain of the evaluation set. Section 5.5 contains a summary about the entire chapter.

5.1 | Evaluation Strategy

As mentioned in Section 3.4.2.2, we had 2 collections of corrected document pairs at our disposal (Vella, 2015, 2022). The first collection, comprised mostly of Word documents, was used as training data for our original benchmarks. The second collection was a selection of PDF files. We kept the latter as our evaluation set. The strategy was to recall each of our developed models and apply them to the grammatically incorrect files. The generated (decoded) outputs could then be scored against the gold references, i.e., the grammatically correct files of each pair.

Decoding using Marian is an equally heavy operation, and due to time limitations, we could not apply the translation process to all 105 files. Instead, we decided to extract

a smaller sample of files on which we could validate our existing models. We then cross-examined the results of these models against a final model, which was trained on the remaining files and included all the adaptations examined in the previous chapter.

The dataset comprised different correction exercises by diploma students. These exercises were organised in small groups of excerpts from different literary sources, mainly consisting of local works of prose. It is crucial to remember that this set of model tests was completely blind, and the tests were being executed against data for which the models had not been trained. Even the domain was completely new, as all the previous datasets, including Common Voice and the MLRS corpus, did not contain any observations akin to local literary works. We used a single group of excerpts of 13 files, totalling nearly 500 sentence pairs.

5.1.1 | ERRANT Scorer

To achieve our last objective, we decided to use the same scorer that was used in the BEA-2019 shared task, called ERRANT (Bryant et al., 2019). ERRANT is a scorer that was originally based on an earlier scorer named M^2 (MaxMatch). The original M^2 was created to address the ambiguities between source strings and hypothesised corrections by evaluating according to the edit operations, which should make the source input sentence match the standard gold reference as much as possible. ERRANT was an improvement over M^2. For example, M^2 could only evaluate span-based corrections (the entire input sequence versus the reference sequence), but ERRANT also supported token-based evaluation. Additionally, ERRANT could report about error-types if the source files had the required annotations.

ERRANT was conceived by Bryant et al. (2017) and was explicitly designed to evaluate sets of sequence pairs. ERRANT was also dataset-agnostic, so it did not require any training to recognise the source files. However, as mentioned before, ERRANT needed an annotation scheme for identifying error types, and the annotation scheme was language specific. We did not possess a compatible annotation scheme for Maltese, so we investigated the impact this limitation would have on ERRANT. Following some experiments, we determined that error-type recognition was the only level of evaluation that depended on annotation tagging, so this part of the evaluation had to either be skipped or done manually. Otherwise, all other supported evaluation methods, span-based and token-based, were still possible.

ERRANT reports results in terms of precision, recall and $F_{0.5}$ whereby $F_{0.5}$ prioritises precision over recall. In order to generate these scores, we needed first to generate *MaxMatch files* (.M2) for tokenised target language reference files. The M2 files were

Table 5.1: Evaluation Scores for token-based error detection.

Model	Token-Based Detection		
	Prec.	Rec.	$F_{0.5}$
Baseline Transformer (Vaswani)	66.39	**40.17**	58.73
Baseline Seq2Seq	90.78	32.62	66.91
Baseline AMUN	89.55	30.79	64.81
Baseline + Source Corruption	83.08	38.52	67.47
Baseline + Tied Embeddings	**91.26**	33.18	67.60
Baseline + Pretrained BERTu	85.04	31.71	63.63
Baseline + Pretrained mBERTu	87.92	32.36	65.45
Domain Adapted	93.37	32.68	**68.08**
Domain Adapted + Large Vocabularies	90.16	33.97	67.75
Domain Adapted + Tied Embeddings	84.93	31.98	63.80
Domain Adapted + Pretrained BERTu	87.04	32.18	64.91
Domain Adapted + Pretrained mBERTu	80.27	32.11	61.75
Final	**90.24**	**22.43**	**56.23**

formatted in a pattern where each sentence was prefixed with the letter *S*, followed by all the relevant annotations prefixed with the letter *A*. The annotations described each token in the source and target files about the phrase type and which index positions they occupy. For example, the first word in the Maltese phrase, 'Jiena nixmi lura' (translates to 'I walk backwards'), would be tagged as a noun phrase occupying index position 0 to index 1. During evaluation scoring, ERRANT would generate a new M2 file for the source language phrases, annotated with all the necessary amendments to regenerate the original reference sequence.

5.2 | Analysis

The evaluation scores for the blind tests are depicted in Table 5.1, Table 5.2 and Table 5.3. We observed a standard behaviour across all models: diminishing scores from token-based analysis (Table 5.1) to span-based analysis (Table 5.2) and once more when moving from error detection to error correction (Table 5.3). In the previous chapter, we could not determine the most robust baseline architecture. However, the differences across the Baseline models was more pronounced this time, indicating that the AMUN transformer implementation was a stronger contender for machine translation. This observation was consistent with the literature's interpretation of transformers being more advantageous when dealing with OOV words and unforeseen data.

Table 5.2: Evaluation Scores for span-based error detection.

Model	Span-Based Detection		
	Prec.	Rec.	$F_{0.5}$
Baseline Transformer (Vaswani)	64.05	27.30	50.46
Baseline Seq2Seq	73.81	26.73	54.58
Baseline AMUN	72.69	25.97	53.46
Baseline + Source Corruption	69.43	**28.49**	53.93
Baseline + Tied Embeddings	79.43	26.83	57.06
Baseline + Pretrained BERTu	81.48	26.15	57.25
Baseline + Pretrained mBERTu	78.94	25.59	**55.71**
Domain Adapted	**83.02**	25.08	56.78
Domain Adapted + Large Vocabularies	81.89	25.28	56.56
Domain Adapted + Tied Embeddings	80.17	24.18	54.79
Domain Adapted + Pretrained BERTu	80.06	24.06	54.63
Domain Adapted + Pretrained mBERTu	79.73	23.65	54.08
Final	**82.08**	**16.94**	**46.40**

Table 5.3: Evaluation Scores for error correction.

Model	Correction		
	Prec.	Rec.	$F_{0.5}$
Baseline Transformer (Vaswani)	20.18	08.60	15.90
Baseline Seq2Seq	22.78	08.25	16.84
Baseline AMUN	24.21	08.65	17.81
Baseline + Source Corruption	21.86	08.97	16.98
Baseline + Tied Embeddings	25.69	08.68	18.46
Baseline + Pretrained BERTu	27.16	08.72	19.08
Baseline + Pretrained mBERTu	26.52	08.59	18.71
Domain Adapted	**31.36**	09.47	21.45
Domain Adapted + Large Vocabularies	31.12	**09.60**	**21.49**
Domain Adapted + Tied Embeddings	30.60	09.23	20.92
Domain Adapted + Pretrained BERTu	28.94	08.70	19.75
Domain Adapted + Pretrained mBERTu	30.81	09.14	20.90
Final	**56.32**	**11.62**	**31.84**

5.2.1 | Baseline Scores

The scores for the baseline models are prefixed with *Baseline* in Table 5.3. The baseline AMUN transformer outperformed the baseline Seq2Seq model by an $F_{0.5}$ score margin of approximately 1% during error correction (ERRANT scores only span-based error correction). The Seq2Seq model outperformed the traditional Vaswani transformer by a slightly smaller margin. Keep in mind that the Vaswani transformer was one of the first EncDec models that used an attention mechanism. It had not been optimised since 2017, and the difference between it and the Nematus AMUN implementation was barely noticeable but still present.

5.2.2 | Synthesised Data

In Subsection 4.1.3, questions were raised regarding using synthesised data during the Domain & Error Adaptation experiment. The confusion was because the inclusion of the synthesised sets had diminished the BLEU score of the model. We suspected that the phenomenon could have occurred because of the increase in the size of the training and testing sets. The oversight could have created an inconsistency in the conditions set by the experimentation process. During the evaluation, all the adaptation models were also tested. This time around, using entirely new data, the domain-adapted model outperformed the baselines in span-based error detection and span-based correction by approximately 4%.

These results inferred that introducing synthesised data could have a double-edged effect. It dampened the model's quality when scored against in-domain examples but increased its quality when scored against out-of-domain examples. We believe that if a GEC model was to be applied to correct texts in its training domain, introducing synthesised data from other domains could work against the model's ability to specialise for its target area. On the other hand, introducing synthesised data could make the model better equipped for handling unforeseen observations. While its precision might suffer in one domain, it would open up the solution to better cater to other domains, thus resulting in a more generalised model.

5.2.3 | Source Word Corruption

Again, we observed that the purposeful corruption of source data via removing tokens from the datasets did not help the model. The source corruption exercise (see *Baseline + Source Corruption* in Table 5.3) had been executed using a baseline AMUN model. The $F_{0.5}$ dropped by a small margin. Ironically, the recall scores for this particular experi-

ment were stronger than those of the other baseline models. However, considering this adaptation involves the literal removal of tokens from the source files, it stands to reason that this measure could lower the potential to pick up false negatives during evaluation.

5.2.4 | Large Vocabularies

Surprisingly, the scores between the domain-adapted and the large vocabulary models were quite neck-and-neck. In the previous chapter, we determined that large vocabularies could lower the quality of a correction model. However, we noticed that the larger vocabulary model had obtained a slightly higher score than the standard domain-adapted model in these last results. We suspected that this improvement was because the larger vocabularies might have lowered the number of OOV instances.

5.2.5 | Adaptations

We observed a steadily declining trend in all models' span-based error detection and correction after domain adaptation. This finding again conflicted with the heightening BLEU scores we observed during the models' training. Following domain adaptation, all vocabularies had been locked to a fixed size of 22,000 entries as this was a requirement for full-length tied embeddings. We could therefore rule out any issues caused by expanding vocabulary size.

Tied embedded and pre-trained models were registering a slight deterioration. Notably, and somewhat surprisingly, the previously identified best-performing model, i.e. pre-trained model with BERTu, registered the weakest $F_{0.5}$ score out of all the adaptations. However, it was still better than the baseline models. Tied embeddings and pre-training all save the similar end goal of predisposing the model to specific outputs rather than others, so we believe that this is why the scores suffered. The adaptations created an over-fitting issue, and the models ended up suffering when doing corrections in new domain territory.

5.3 | Issues

We looked deeper into the actual generated correction files to better adjudicate the system's output. Several issues were discovered, many of which were not immediately discernible just by relying on the $F_{0.5}$ scores.

5.3.1 | Translation Behaviour

The first immediately apparent characteristic was how the solution acted as a translation task. This behaviour was not unexpected since all GEC solutions based on NMT were essentially translation tasks. However, it was evident that the solution did not truly *comprehend* what it was meant to be doing. There were numerous instances where grammatically correct sentences were being erroneously translated, and this discovery correlated to the high number of false negatives picked up across all the models. This kind of issue would not have been as big a problem had the task indeed been a translation from one language to another. There could be several good ways to express a Maltese statement in English. For example, the Maltese phrase 'Għandi bżonn raqda tajba' literally translates to 'I need a good sleep'. Even though that is the *best* translation, other translations could have also sufficed, such as 'I need a good nap' or 'I need a proper rest'.

Languages enjoy broad vocabularies that accommodate broader ranges of semantics. Either of the overhead phrases could potentially be accepted as an accurate translation by a speaker. This characteristic does not carry over to Grammar Error Correction. If the target sentence deviates too much from the source sentence, it would still be interpreted as an invalid correction even if it was grammatically correct and semantically cohesive.

There is only one accurate correction for 'Andi bżonn raqda tajba', which is to add the silent consonant *għ* at the beginning of the sentence. In one such instance, the GEC tool erroneously corrected the phrase 'kollox pulit' ('everything is neat') to 'kollox blordni' ('everything is in order'). The two phrases are similar and might be passable translations for a human reader, but they are not proper corrections.

5.3.2 | False Negatives

The solution suffered from immensely low recall. Beyond the problems associated with invalid corrections, an ideal GEC solution should not undo correctly written grammar. The behaviour of unearthing correct phrases comes from the fact that the entire GEC solution was trained in an unsupervised way. In the absence of any tagging information, there is nothing that could inherently guide the trained model into distinguishing incorrect grammar from correct grammar.

When the tool suggests a correction unnecessarily is called a false negative. For example, the system incorrectly changed the phrase 'kultant' to 'kull tant' in the phrase 'kultant kien iressqu lejn xofftejh' which translates to 'he would occasionally bring it closer to his lips'. In Maltese, both phrases are technically valid. The former, 'kultant',

roughly translates to ‘every once in a while’ or ‘occasionally’. The latter, ‘kull tant’, translates similarly but requires the object of the phrase to be specified. For example, ‘kull tant żmien’ roughly translates to ‘every such length of time’. In the context of the example phrase, the former is correct.

Figure 5.1 shows the distribution of true positives, false positives and false negatives across all of our evaluation experiments. We concluded that upon attempting to use the solution on unforeseen data, the solution found it challenging to determine what needed fixing and flagged many more words for correction than it was meant. Most information was not accounted for during training; therefore, the solution treated these instances as errors.

5.3.3 | Domain Bias

Since the solution behaved like a translator, there was considerable domain bias. The GEC tool was trained on three primary sources prior to evaluation. Common Voice was mostly comprised of random statements. Meanwhile, the original batch of corrected documents mainly was based on political discourse. The varying sources arising from MLRS had a wider variety of subject matter, including academia, culture, European and Maltese law, news and opinions, parliamentary debates, religion and sports. It was not an equal distribution at all. The news and political material were much more extensive than the other sections and accounted for 86% of the entire dataset.

This factor caused the GEC tool to perform corrections and inadvertently change the sentence’s meaning. In several cases, the tool shifted the source sentence’s subject matter to something closer to the data on which it was trained. In one instance, the phrase ‘Il-ġurament li ħalef fil-maqdes tal-Mulej’ (‘The testament he swore upon in the temple of God’) was translated to ‘Il-mistoqsija li saret fil-każ tal-Mulej’ (‘The question that was asked in the case of God’). Examples like these littered the decoded files, indicating that the solution experienced immense difficulty separating itself from its training material’s domain.

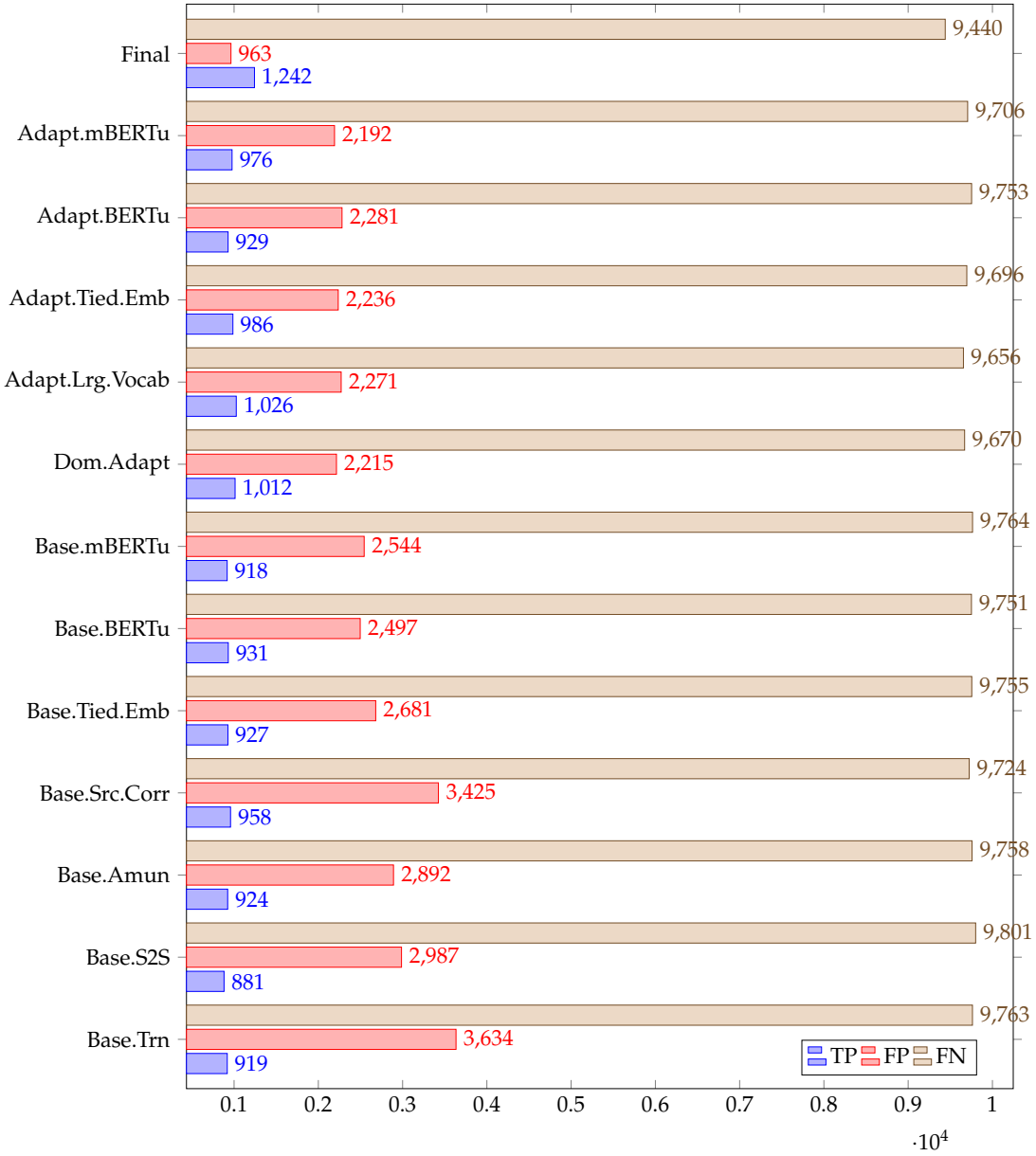

Figure 5.1: Evaluation Outcomes

5.3.4 | Sentence Structure

Generally, all the models behaved better on shorter sentences than longer ones. Most sequences translated correctly were shorter phrases, typically six tokens or less. This sort of issue might have been circumvented better by modifying the size of the beam search window during training. An additional issue was that sometimes the tool would repeat the same phrase repeatedly. This phenomenon is called *Neural Text Degeneration* (Holtzman et al., 2020). It seems to be a by-product of beam search where extra-long sentences earn higher probability scores. The beam search thus gets stuck in a long repetitive loop and delivers a long incoherent sequence. Holtzman et al. (2020) recommends some statistical options to fix the problem. However, none of the methods had ever been mentioned alongside the state-of-the-art, so they were not attempted.

It was also observed that the models with smaller vocabularies, particularly the baselines, frequently could not reproduce certain words in the output. This detail was in contrast to the models with larger vocabularies, and most notably, the ones pre-trained on the BERT language models rarely ran into this issue.

5.3.5 | Inter-Rater Reliability Testing

Inter-rater reliability refers to the extent to which different observers of the same phenomenon agree with their results. The agreement does not refer to achieving identical scores, but exhibiting a strong resemblance and homogeneity in the pattern of the results. This can be measured using different statistical techniques and is used to establish the validity of the observers. For example, the Fleiss Kappa measure can be used in the case of nominal data and the Kendalls Tau measure can be used for ordinal data. In our case, the results of our experiments were all expressed in metric variables (true positives, false positives and false negatives), therefore the best option for measuring inter-rater reliability was using the Intraclass Correlation Coefficient (ICC) (Liljequist et al., 2019).

The observers, referred to as 'raters', are each individual model conceived during the experimentation phase (as detailed in Tables 5.1, 5.2 and 5.3). The 'subjects' in our case are the evaluation files. We performed ANOVA (Analysis of Variance) on the measured data matrix, calculating the various possible sums of squares and the mean squares (MS). ANOVA is used to discover the statistical significance of the differences between different groups.

Intraclass correlation is expressed in Equation 5.4. $var(\beta)$ is the variability due to differences in the subjects. This refers to the differences between the observed files. $var(\epsilon)$

Table 5.4: Intraclass Correlation Coefficients

Measure	ICC	Reliability
True Positives	83.56%	Good
False Positives	54.02%	Moderate
False Negatives	96.78%	Excellent

is the variability due to differences in the evaluations of the subjects by the observers. For example, one model discovers more errors in one sequence than another model. $var(\alpha)$ is the variability due to differences in the rating scale used by the observers. For example, some of the models interpret certain tokens to be correct while others do not.

$$var(\beta) = (MS_{Row} - MS_E)/k \tag{5.1}$$

$$var(\epsilon) = MS_E \tag{5.2}$$

$$var(\alpha) = (MS_{Col} - MS_E)/n \tag{5.3}$$

$$ICC = \frac{var(\beta)}{var(\alpha) + var(\beta) + var(\epsilon)} \tag{5.4}$$

In the aforementioned equations, n refers to the number of subjects (rows) and k is the number of raters (columns). MS is the mean-square function and E refers to the measurement error. The final ICC calculation is expressed in Equation 5.5.

$$ICC = \frac{MS_{Row} - MS_E}{MS_{Row} + df_{Col}MS_E + (df_{Col} + 1)(MS_{Col} - MS_E)/(df_{Row} + 1)} \tag{5.5}$$

We applied the ICC formula for the results obtained by our models and extracted a score computed over the true positive, false positive and false negative counts of all the models. The resulting values are illustrated in Table 5.4. The models exhibit good reliability for true positives (83.56%) and excellent reliability for false negatives (96.78%). The lowest score is that of false positives, but 54.02% is still considered moderately reliable (Koo and Li, 2016).

5.4 | The Final Model

After concluding the blind tests, we ran one last exercise that included training data from the domain of the evaluation sets. As mentioned earlier, we evaluated against a sample of files from 105 corrected document pairs. We took the best-performing model that used synthesised data, tied embeddings and pre-training from the monolingual model identified from Chapter 4. We executed the training process for all the data we had used and included the 105 additional training files. Like all the experiments conducted during this study, we capped training at 20,000 batch iterations.

5.4.1 | Training Results

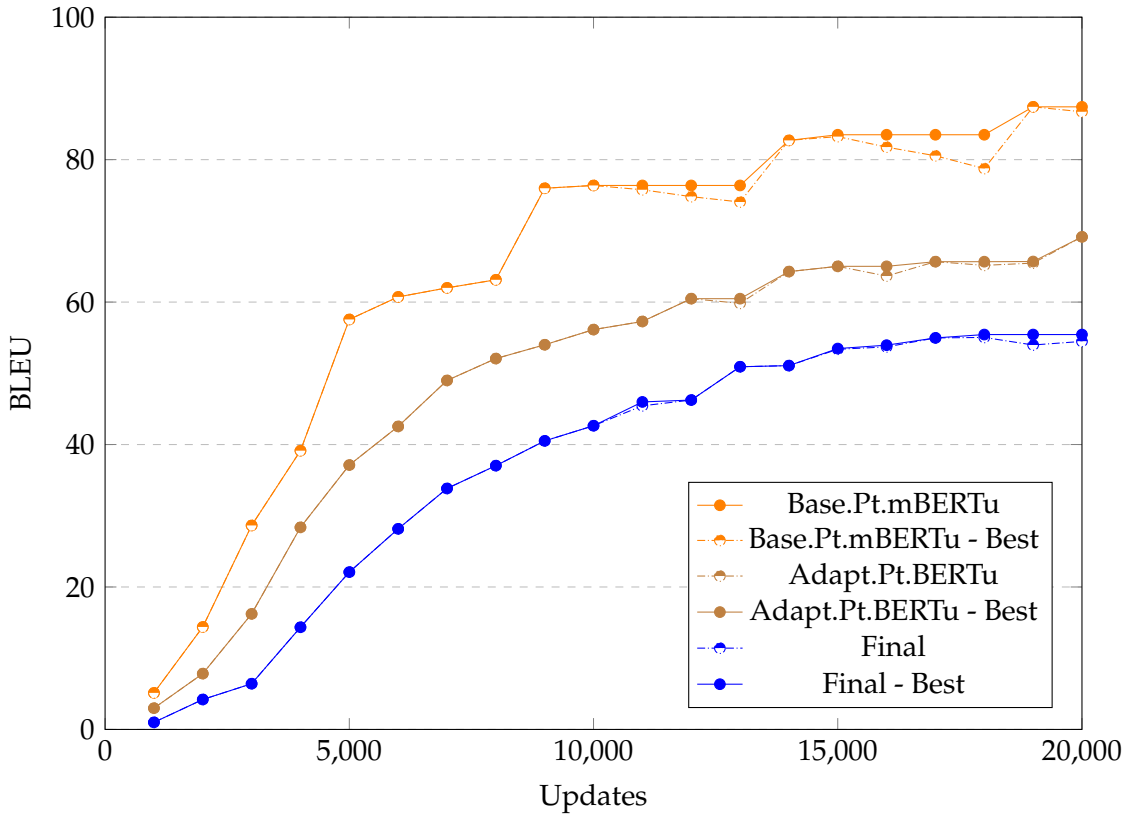

Figure 5.2: Final Model

The results can be seen in Figure 5.2. The final model achieved a BLEU score of 55.44%. It under-performed compared to the previously established top-scoring baseline and adapted models. However, we expected this behaviour to repeat itself after it occurred the first time in Subsection 4.1.3 and it did not necessarily mean that the model would

be a poorer option for the evaluation test. In fact, when tested against the evaluation set, it achieved an $F_{0.5}$ score of 31.84%. Therefore, it registered an improvement of approximately 10% over the best-performing models built during the last experimentation phase.

We looked into the correction quality of the tool. Table 5.5 shows 10 examples of the tool making valid corrections. Table 5.6 shows a further 10 examples of where the tool made mistakes.

5.4.2 | Valid Corrections

In Table 5.5, we can see that the tool applied necessary spelling corrections such as in Example 1 where the original sentence mistakenly included a 'j' in the intended word 'idea'. This was a non-word phonetic error due to the similar pronunciations of 'eja' and 'ea' in Maltese. Another example is Example 9 which occurs due to the phonetic similarity between the Maltese vowels 'i' and 'ie'.

In Example 8, the original sentence omitted a character from a double letter 'bb' in 'impossibbli'. In Example 5, we observe the opposite where a letter was inserted erroneously to create a double letter. In the case of Example 4, we can observe an instance of a real-world phonetic homonym error. It is a case of confusion between the word 'kolla' (that means 'glue') and 'kollha' (that means 'all'). In Maltese, both these words are pronounced exactly the same.

Examples 2, 6, 7 and 10 are all cognitive errors. Example 2 is a misunderstanding of when to use the determiner 'mall-' versus 'mal-'. We see a similar effect in Example 10. The determiner 'għar-' would typically be valid in cases where the subsequent word would start with the consonant 'r', but 'rasu' is an exception to this rule.

In the case of Example 6, the error is due to using an obsolete Maltese grammatical rule that is no longer accepted i.e. substituting the word 'u' (which means 'and') with 'w' when the subsequent word starts with a vowel. In Example 7, we observe a similar idea where the preceding 'i' in 'inħossni' was not present in the original sentence. This would have made sense if the word immediately before it had ended in a vowel, but in this case, it did not.

Examples 2, 7 and 10 are all real-world cognitive errors. Since Example 6 is no longer considered acceptable Maltese grammar, it is thus categorised as a non-word cognitive error. Example 3 is a classic non-word typographic error where the writer most probably forgot to write the initial 'i' in the word 'iżda'.

Table 5.5: Examples of Valid Corrections.

No	Original Sentence	Corrected Sentence
1.	**L-ideja** li joqgħod għal ftit żmien waħdu	**L-idea** li joqgħod għal ftit żmien waħdu
2.	Il-karozza **mall-bankina**	Il-karozza **mal-bankina**
3.	**Żda** huma lil ħadd ma kienu	**Iżda** huma lil ħadd ma kienu
4.	Mhux talli l-istudenti ngħaqdu **kolla** flimkien	Mhux talli l-istudenti ngħaqdu **kollha** flimkien
5.	Qed inħoss biss sturdament stramb u **għejja** kbira	Qed inħoss biss sturdament stramb u **għeja** kbira
6.	F'isem ħuti **w** ommi	F'isem ħuti **u** ommi
7.	Qed **nħossni** qisni se norqod iżda ma rridx	Qed **inħossni** qisni se norqod iżda ma rridx
8.	Imma **possibli** mietet? U le, kif jista' jkun?	Imma **possibbli** mietet? U le, kif jista' jkun?
9.	Wara ħames xhur xeba' jisma' l-leħen **irqieq** tagħha	Wara ħames xhur xeba' jisma' l-leħen **irqiq** tagħha
10.	Qalilhom li se jippjana xi ħaġa **għar-rasu** iżda kien jaf li ma kien se jagħmel xejn	Qalilhom li se jippjana xi ħaġa **għal rasu** iżda kien jaf li ma kien se jagħmel xejn

5.4.3 | Invalid Corrections

In Table 5.6, we can observe several examples where the tool failed to arrive at a valid correction. In the majority of the cases, these were instances when the tool mistook valid words to be errors and corrected them unnecessarily.

Example 1 shows an instance when the tool incorrectly changed the spelling of 'tlett' to 'tliet'. Both spellings are valid Maltese words and both are used to represent a count of 'three'. However, the original sentence correctly uses the modified spelling that is required when the subsequent word begins with the vowel 'i'. Similarly, in Example 3, the hyphen in the determiner of the original sentence was indeed correct. The circumstances for using this spelling are rarer in Maltese, which explains why the tool had a hard time interpreting the sentence as correct. In Example 4, the difference between the original and corrected sentence is rooted in context but both exist in Maltese. In this example though, the original sentence was the correct one.

Table 5.6: Examples of Invalid Corrections.

No	Original Sentence	Corrected Sentence
1.	Imma b'familja ta' **tlett** itfal ma setax jagħmel li jrid	Imma b'familja ta' **tliet** itfal ma setax jagħmel li jrid
2.	Fil-bidu tas-sena kien hemm tlieta oħra **bħalhom**	Fil-bidu tas-sena kien hemm tlieta oħra **bħal**
3.	Kemm-**il** darba	Kemm **il-**darba
4.	**Dil-biċċa** ġieli kienet iddum	**Din biċċa** ġieli kienet iddum
5.	Imma issa, li **z-**zija Anna kienet beżqitha	Imma issa, li **ż-**zija Anna kienet beżqitha
6.	**Kultant** niskanta bija nnifsi	**Kull tant** niskanta bija nnifsi
7.	Għal Milan **ikun** hemm titjira kull siegħa	Għal Milan **jkun** hemm titjira kull siegħa
8.	L-uffiċċju **għab** u l-kamra nbidlet f'ġenna	L-uffiċċju **nies** u l-kamra nbidlet f'ġenna
9.	**Xhin** toqgħod taħseb kif il-ġrajjiet	**X'ħin** toqgħod taħseb kif il-ġrajjiet
10.	Wenzu kien **ġieb** miegħu flixkun	Wenzu kien **iġib** miegħu flixkun

There were instances when the tool made nonsensical mistakes such as in Examples 2 and 8. In Example 2, the tool erroneously inflected the word 'bħal' even though it was already valid. In the case of Example 8, the replacement word made absolutely no sense. we suspect that this happened simply due to an abundance of examples in the training data of the words 'uffiċju nies' ('an office of people'). In the meantime, the tool had no training examples using the verb 'għeb', which led it to consider the word as an error.

5.4.4 | Comparison with BEA-2019

Table 5.7 places our solution among the rankings of the BEA-2019 shared task. The other systems' results were referenced from the report of Bryant et al. (2019). Before analysing the numbers, it is crucial to remember that our solution was evaluated against a different dataset since we were using a different language, and the size of our evaluation sets also varied. We did not input a rank number for our system, as there were too many varying parameters to justify claiming that our solution outperformed any of the ones

Table 5.7: BEA-2019 Shared Task

Rank	Teams	TP	FP	FN	P	R	$F_{0.5}$
1	UEDIN-MS	2312	982	2506	70.19	47.99	64.24
2	Kakao&Brain	2412	1413	2797	63.06	46.30	58.80
3	LAIX	1443	884	3175	62.01	31.25	51.81
4	CAMB-CUED	1814	1450	2956	55.58	38.03	50.88
5	UFAL	1245	1222	2993	50.47	29.38	44.13
6	Sitcimprove	1299	1619	3199	44.52	28.88	40.17
7	WebSpellChecker	2363	3719	3031	38.85	43.81	39.75
-	**Maltese GEC**	**1241**	**963**	**9440**	**56.32**	**11.62**	**31.84**
8	TMU	1638	4314	3486	27.52	31.97	28.31
9	Buffalo	446	1243	3556	26.41	11.14	20. 73

mentioned in the shared task.

With that in mind, we were still satisfied with the overall result of our model. We noticed that our solution performed very well in terms of model precision. It faltered when it came to recall. The poor recall is attributed to the abnormally high number of false negatives. Since the number of false negatives was high, our GEC solution was over-correcting. In the context of the NMT paradigm, over-correcting means that the correct source words were not yielding the same token after passing through the layers of the RNN.

5.5 | Summary

We evaluated all the models generated in Chapter 4 against new datasets in this chapter. The datasets comprised of never-seen-before corrected document pairs. We evaluated against a subset of files and discovered that the models that used synthesised sets outperformed those that did not. However, additional adaptations did not bear the same benefits as expected when applying the models without training.

We trained a final model on datasets closer to the evaluation sets and managed to improve the scores. The score was 31.84%. Compared to the rest of the submissions on the BEA-2019 shared task, our model exhibited good precision but poor recall.

6

Conclusions

This section discusses the conclusions drawn from the undertaking of this study. This chapter is divided as follows. Section 6.1 revisits each objective that was defined back in Chapter 1. Section 6.2 elaborates in detail about the system's shortcomings. Section 6.3 describes potential future work that could build on top of this solution. In Section 6.4, we close off this dissertation with our own final contemplation.

6.1 | Revisiting the Aims and Objectives

At the beginning of this study, we defined three distinct objectives.

6.1.1 | First Objective

Evaluate existing state-of-the-art deep-learning methods for grammatical error correction and implement a baseline model using a neural-based approach.

Through an exhaustive literature review, we identified the potential of neural-based methods as an alternative to statistical systems and a superior successor entirely. The Encoder-Decoder model was the most prominent architecture used in Neural Machine Translation. Its application was not only limited to traditional cross-lingual translation but also for Grammar Error Correction.

We became acquainted with and eventually increased our command of the Marian NMT toolkit to achieve this objective. The toolkit's prestige was well merited, having been built on efficient principles but having also implemented some of the best performing architectures in GEC as a whole, such as the renowned Nematus transformer,

the Vaswani transformer and their forerunner, the Sequence-to-Sequence model. Furthermore, several studies that achieved state-of-the-art results had either sought to outperform Marian or based their work on top of the toolkit. Marian NMT was among the top quality NMT platforms recently and had occupied the top spot in the Low Resource track of the BEA-2019 shared task.

We thus implemented three distinct models based on the Nematus as mentioned earlier, Vaswani and Seq2Seq architectures. We trained them using a base dataset comprising 4000 authentic sentence pairs, whereby each grammatically incorrect sentence in the source file aligned its grammatically correct counterpart in the target file. We trained these baseline models for 20,000 iterations and arrived to the strong BLEU scores of 78.11%, 77.79% and 78.36% for the Nematus transformer (a.k.a AMUN), the Vaswani transformer and the Seq2Seq model respectively. All three of these scores were very positive and compared quite favourably against several systems identified in the literature. Despite its slightly lower score, we decided to champion the AMUN architecture due to its smaller hardware impact. However, either of the other alternative EncDec models was still a worthy baseline system in their own right.

We believe we achieved this first objective fully by delivering these high-quality implementations that achieved high scores.

6.1.2 | Second Objective

Augment the Grammar Error Correction model using supported techniques that are meant for resource-limited settings.

Having established our baseline models, we then turned our attention to groundbreaking techniques for enhancing model quality in low-resource settings. Chief among these was the technique of appending synthesised data to the model's training sets. This approach allows researchers to artificially expand the sizes of their data in order to counteract the negative impacts of having too few observations for unsupervised training.

We built our error synthesiser that could inject errors of omission, insertion, transposition, keyboard proximity, article errors and font errors into texts. We artificially expanded the data from 4000 rows to 13,000 rows. The influx of additional data impacted the scores of the models, partly because the train-test split increased as well. However, during evaluation, we observed a consistent improvement in the models that had undergone error synthesis. We concluded that synthesised data could make a better-generalised model while keeping the training data without synthesised instances would leave one with a domain-specialised model better suited for smaller scopes.

Based on the work of Junczys-Dowmunt et al. (2018b), we applied several other adaptations, including source word corruption, tied embeddings and pretraining. Among these, we realised that tied embeddings increased the scores of our baseline models. For pretraining, we leveraged pre-existing BERT models trained on Maltese. Both the monolingual and the multilingual variants increased the scores of our models. These same experiments were repeated even on the models with the added synthesised data and registered the same improvements.

Having demonstrated how the application of adaptations benefited the quality scores of our GEC models, we believe that we have also achieved this objective.

6.1.3 | Third Objective

Apply the model in the context of the Maltese language and evaluate it using a similar methodology to other related neural-based systems in environments with resource scarcity.

All the training data fed into our models were written in Maltese. We could think of no better way to evaluate our models than to follow the same evaluation practices adopted during the BEA-2019 shared task. The BEA-2019 shared task was the crucible of several state-of-the-art systems explicitly designed for low resource GEC. All the systems that participated in the shared task had to use the same scorer, ERRANT. By evaluating our solution using the same scorer, we could produce results that could be cross-examined against the actual rankings of the shared task itself.

Since we were dealing with a completely different language, we could not evaluate against the WI+LOCNESS dataset like the actual BEA-2019 participants. As an alternative, we decided to introduce new data into our experimentation. The new data comprised the latter half of our corrected document pairs. We had kept this data purposely for evaluation, and none of the models had been trained initially on it. We used the Marian decoder functionality to recall all of our models and executed them against our new datasets.

The results of the blind translations were not the most satisfactory. We, therefore, retrained a new model to gauge its effectiveness if it had trained on some of the new data. The results were better the second time, and we achieved an $F_{0.5}$ score of 31.84%. We gauged how our tool fared when compared against the other officially ranked participants of the Low Resource Track in the BEA-2019 shared task.

Having adopted the very same scorer (ERRANT) as was used in the Low Resource track of the BEA-2019 shared task, we guaranteed that our evaluation process would be

comparable to the evaluation methodologies of related low resource GECs. Due to this, we believe that we fulfilled our third objective.

6.2 | Critique and Limitations

We believe that the GEC solution described in this study was a solid attempt at resolving a long-standing issue in Maltese computer-based language resources. However, it is not without its fair share of flaws.

6.2.1 | Critique

The system experienced what we referred to as *translation bias*. By treating error correction as a translation task, the GEC tool exerted a certain degree of liberty with its solutions. It suffered very high rates of false negatives, often attempting to correct errors that were not even there. Furthermore, the system also experienced a lot of domain bias. It seemed that no matter the context of the target texts, our solution's translations often moved the goalposts of the subject domain back to something political or news-related, as these happened to be the lion's share of the training sets' subject matter.

Despite relying on deep learning paradigms, the system suffered from OOV issues. While we think that the OOV problems could have been worse, we did not manage to eliminate the issue.

6.2.2 | Limitations

It was immediately evident that the data we had in our possession for training was far from pristine, which was not very reassuring considering that many of our training instances had been sourced from the MLRS server. We had naively expected that the data would be grammatically coherent, but our sources were rife with grammatical issues, even in the supposed *valid* datasets. We spent much time repairing these datasets, and we are sure we missed out on some essential fixes.

The data was also full of garbage text, including repeated hyphens, underscores and other symbols. There were names, pronouns and other confusing titles. Most aggravatingly, many observations were written in English. These were removed, but we might have missed a number of these while curating the data again. As far as curating was concerned, we did our best to automate as much of the process as possible. However, most issues in our source texts lacked any cohesive pattern we could adequately target with an algorithm or a regular expression. These issues had to be removed manually.

No matter how many tricks we could have pulled off with synthesis, nothing could be a perfect substitute for real-world data. We believe that we might have been overly reliant on the synthesis due to our circumstances. Our models lacked much real-world training that could have guided them better towards identifying the actual errors in the text and possibly even reducing the number of false negatives. We did not have enough examples to produce a heuristic that could guide our noising strategies when we conducted error synthesis. The most prevalent errors were missing fonts. It would have been better if we had different percentage scores for different error types so we could guide the seeding parameters of our error synthesis tool.

Lastly, we hoped we could have had more time to do more exhaustive testing. We wonder how the solution would have behaved with different test-train splits, tweaked learning rates and optimiser rates. We wonder whether we could have experimented further with the size of the beam search mechanism or if tweaking the dropout rates of the source, target, and RNN layers would have changed our results. We were training extensive models and could not possibly cover all our bases in the allotted time.

6.3 | Future Work

We strongly believe in the potential for this solution to be expanded in the future. Firstly, we know that the system's performance could be increased if there was cleaner data for training. Even if we assume the use of the same datasets as the ones used for this study, the solution scores will increase if they are more meticulously vetted and thoroughly sanitised. If additional data were added to this pipeline, it would do well to be sourced from a broader range of domains. It would improve the system if the data coming from those different domains amounted to roughly the same size as not to bias the GEC tool in favour of one domain over the other. An interesting opportunity would be to contact local authors and obtain the source texts of their books and articles. Literary works, newspaper articles and online articles are very likely to have undergone an editing phase to iron out grammatical errors. The pre-editing and post-editing files would be a great addition to the pool of data sources that support a system such as the one described in this study.

As mentioned in Subsection 6.2.2, the presence of a heuristic model that could guide synthesis would open up possibilities for the tool. The information coming from this model could be used to guide error synthesis rather than have it randomised or based on limited information. Furthermore, the synthesis tool could be expanded further to cater to more error types not handled by this study. For example, there could be synthesis

strategies for phonetic errors, incorrect pronouns, incorrect use of the Maltese vowel *ie* and common incorrectly typed double letters. In conjunction with the heuristic model, the application of these new synthesis strategies will become much easier and much more useful. The trained GEC model will always perform better if its observations are coming from real-world data sources. If those are unavailable, it stands to reason that the next best thing is to synthesise errors based on error patterns that have appeared in real-world scenarios.

Similarly to the literature, one could employ backward translation as a form of synthesis. Backward translation works by training an NMT system to work the other way around and translate a valid text into an invalid text with common grammatical mistakes. This method can be used either in conjunction with or as a replacement for a heuristic model. The resulting error-synthesising trained model can then be applied to numerous texts to create elaborate synthesised sets. In the case of the Maltese language, it is far rarer to find error information than it is to find text sources, so this technique could prove to create the most dramatic improvements in the quality of the system. To bolster the pool of training data sources, future researchers could train an OCR to read text written in Maltese and apply the OCR to as many Maltese literary sources as they can find. Subsequently, the resulting datasets can be passed to the error synthesiser.

In Chapter 5, we discussed the limitations that we encountered with the ERRANT scorer. Whether future researchers continue building on our solution or start building new ones, we strongly believe in the potential of the ERRANT scorer as an evaluation tool. It would be a positive endeavour to train an annotation model and write a rule set that could be integrated with ERRANT. The scorer already supports this, and it would give researchers a means of quickly evaluating document pairs and analysing their corrections down to error-type granularity. This information would give useful insights on which error types our focus should be when creating future GEC tools.

6.4 | Final Remarks

With a final $F_{0.5}$ score of 31.84%, we believe it would be amiss to call our GEC tool a complete solution. We still fell short of a neural-based model that could be robust enough to be deployed in the real world. However, considering the project's novelty, we are still satisfied to have achieved this result. Considering the ample opportunities for improving our system discussed in Section 6.3, we have no doubt that the results of deep-learning based Grammar Error Correction are bound to improve in the foreseeable future.

We hope this work could assist the few researchers who work year-in and year-out to increase the collection of language resources for Maltese. With this study, we leave behind a milestone which could serve as the starting point for future research. We leave behind datasets that have undergone vetting, cleaning and reorganisation so they can be used in future systems. Lastly, we leave behind a small but hopefully useful set of conclusive data that could contribute to future efforts in the field. We are proud of these contributions and hope this study will continue to spur interest in the Maltese language and its available resources.

References

Ahmad, F. and Kondrak, G. Learning a Spelling Error Model from Search Query Logs. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 955–962, Vancouver, British Columbia, Canada, October 2005. Association for Computational Linguistics.

Ahmed, F., Luca, E. W. D., and Nurnberger, A. Revised N-Gram based Automatic Spelling Correction Tool to Improve Retrieval Effectiveness. *Polibits*, 40:39–48, 2009.

Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., Morais, R., Saunders, L., Tyers, F. M., and Weber, G. Common Voice: A Massively-Multilingual Speech Corpus. In *Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020)*, pages 4211–4215, 2020.

Bahdanau, D., Cho, K., and Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate. *ArXiv*, 1409, 09 2014.

Brockett, C., Dolan, W. B., and Gamon, M. Correcting ESL Errors Using Phrasal SMT Techniques. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 249–256, Sydney, Australia, July 2006. Association for Computational Linguistics.

Brown, P. E., Pietra, V. J. D., Pietra, S. A. D., and Mercer, R. L. The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics*, 19:263–311, 01 1993.

Brown, P. F., Cocke, J., Pietra, S. A. D., Pietra, V. J. D., Jelinek, F., Lafferty, J. D., Mercer, R. L., and Roossin, P. S. A Statistical Approach to Machine Translation. *Comput. Linguist.*, 16(2):79–85, jun 1990.

Bryant, C., Felice, M., and Briscoe, T. Automatic Annotation and Evaluation of Error Types for Grammatical Error Correction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 793–805, Vancouver, Canada, July 2017. Association for Computational Linguistics.

Bryant, C., Felice, M., Andersen, Ø. E., and Briscoe, T. The BEA-2019 Shared Task on Grammatical Error Correction. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, Florence, Italy, August 2019. Association for Computational Linguistics.

Casha, R. Spelling MT. `https://spelling.mt/check.html`, 2004. Accessed: 2022-06-01.

Chen, W., Matusov, E., Khadivi, S., and Peter, J.-T. Guided Alignment Training for Topic-Aware Neural Machine Translation. In *Conferences of the Association for Machine Translation in the Americas: MT Researchers' Track*, pages 121–134, Austin, TX, USA, October 28 - November 1 2016. The Association for Machine Translation in the Americas.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Con-*

ference on Empirical Methods in Natural Language Processing (EMNLP), pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

Chodorow, M., Tetreault, J., and Han, N.-R. Detection of Grammatical Errors Involving Prepositions. In *Proceedings of the Fourth ACL-SIGSEM Workshop on Prepositions*, pages 25–30, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

Chollampatt, S., Taghipour, K., and Ng, H. T. Neural Network Translation Models for Grammatical Error Correction. In *IJCAI*, 2016.

Cucerzan, S. and Brill, E. Spelling Correction as an Iterative Process that Exploits the Collective Knowledge of Web Users. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 293–300, Barcelona, Spain, July 2004. Association for Computational Linguistics.

Etoori, P., Chinnakotla, M., and Mamidi, R. Automatic Spelling Correction for Resource-Scarce Languages using Deep Learning. In *Proceedings of ACL 2018, Student Research Workshop*, pages 146–152, Melbourne, Australia, 07 2018. Association for Computational Linguistics.

Felice, M., Yuan, Z., Andersen, Ø. E., Yannakoudakis, H., and Kochmar, E. Grammatical error correction using hybrid systems and type filtering. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 15–24, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

Felice, M., Bryant, C., and Briscoe, T. Automatic Extraction of Learner Errors in ESL Sentences Using Linguistically Enhanced Alignments. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 825–835, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee.

Flachs, S., Stahlberg, F., and Kumar, S. Data Strategies for Low-Resource Grammatical Error Correction. In *Proceedings of the 16th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 117–122. Association for Computational Linguistics, 04 2021.

Gatt, A. and Čéplö, S. Digital corpora and other electronic resources for Maltese. In *In Proceedings of the International Conference on Corpus Linguistics*, pages 96–97, Lancaster, UK: University of Lancaster, 2013. Corpus Linguistics.

Gill, M. S. and Lehal, G. S. Design and Implementation of Punjabi Spell Checker. *International Journal of Systemics, Cybernetics and Informatics*, 01 2007.

Graves, A. Generating Sequences With Recurrent Neural Networks. *ArXiv*, abs/1308.0850, 08 2013.

Grundkiewicz, R. and Junczys-Dowmunt, M. Near Human-Level Performance in Grammatical Error Correction with Hybrid Machine Translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 284–290, New Orleans, Louisiana, 04 2018. Association for Computational Linguistics.

Grundkiewicz, R., Junczys-Dowmunt, M., and Heafield, K. Neural Grammatical Error Correction Systems with Unsupervised Pre-training on Synthetic Data. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 252–263, Florence, Italy, August 2019. Association for Computational Linguistics.

Hochreiter, S. and Schmidhuber, J. Long Short-Term Memory. *Neural Computation*, 9:1735–1780, 11 1997.

Holtzman, A., Buys, J., Du, L., Forbes, M., and Choi, Y. The Curious Case of Neural Text Degeneration. In *International Conference on Learning Representations*, 2020.

Jalili Sabet, M., Dufter, P., Yvon, F., and Schütze, H. SimAlign: High Quality Word Alignments Without Parallel Training Data Using Static and Contextualized Embeddings. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1627–1643, Online, November 2020. Association for Computational Linguistics.

Junczys-Dowmunt, M. and Birch, A. The University of Edinburgh's systems submission to the MT task at IWSLT. In *Proceedings of the 13th International Conference on Spoken Language Translation*, Seattle, Washington D.C, December 8-9 2016. International Workshop on Spoken Language Translation.

Junczys-Dowmunt, M. and Grundkiewicz, R. MS-UEdin Submission to the WMT2018 APE Shared Task: Dual-Source Transformer for Automatic Post-Editing. In *Proceedings of the Third Conference on Machine Translation*, pages 822–826, Brussels, Belgium, 10 2018.

Junczys-Dowmunt, M., Grundkiewicz, R., Dwojak, T., Hoang, H., Heafield, K., Neckermann, T., Seide, F., Germann, U., Fikri Aji, A., Bogoychev, N., Martins, A. F. T., and Birch, A. Marian: Fast Neural Machine Translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia, July 2018a. Association for Computational Linguistics.

Junczys-Dowmunt, M., Grundkiewicz, R., Guha, S., and Heafield, K. 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, June 2018b. Association for Computational Linguistics.

Kalchbrenner, N. and Blunsom, P. Recurrent continuous translation models. *EMNLP 2013 - 2013 Conference on Empirical Methods in Natural Language Processing, Proceeding of the Conference*, 3:1700–1709, 01 2013.

Kaneko, M., Mita, M., Kiyono, S., Suzuki, J., and Inui, K. Encoder-Decoder Models Can Benefit from Pre-trained Masked Language Models in Grammatical Error Correction. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4248–4254, Online, July 2020. Association for Computational Linguistics.

Kashefi, O., Sharifi, M., and Minaie, B. A novel string distance metric for ranking Persian respelling suggestions. *Natural Language Engineering*, 19:259–284, 04 2013.

Kaur, B. and Singh, H. Design and Implementation of HINSPELL-Hindi Spell Checker using Hybrid approach. *International Journal of scientific research and management (IJSRM)*, 3:2058–2061, 02 2015.

Kibrik, A. An Introduction to the Languages of the World. *American Anthropologist*, 101:856–858, 01 2008.

Knight, K. and Chander, I. Automated Postediting of Documents. In *Proceedings of the Twelfth AAAI National Conference on Artificial Intelligence*, page 779–784. AAAI Press, 1994.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., and Herbst, E. Moses: Open Source Toolkit for Statistical Machine Translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

Koo, T. K. and Li, M. Y. A guideline of selecting and reporting intraclass correlation coefficients for reliability research. *Journal of Chiropractic Medicine*, 15(2):155–163, 2016.

Kudo, T. and Richardson, J. 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, November 2018. Association for Computational Linguistics.

Kukich, K. Techniques for Automatically Correcting Words in Text. *ACM Comput. Surv.*, 24(4):377–439, dec 1992. ISSN 0360-0300.

Liljequist, D., Elfving, B., and Roaldsen, K. Intraclass correlation – A discussion and demonstration of basic features. *PLOS ONE*, 14:35, 07 2019.

Luong, T., Sutskever, I., Le, Q., Vinyals, O., and Zaremba, W. Addressing the Rare Word Problem in Neural Machine Translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 11–19, Beijing, China, July 2015. Association for Computational Linguistics.

Micallef, K., Gatt, A., Tanti, M., van der Plas, L., and Borg, C. Pre-training Data Quality and Quantity for a Low-Resource Language: New Corpus and BERT Models for Maltese. In *Proceedings of the 3rd Workshop on Deep Learning for Low-Resource NLP (DeepLo 2022)*, Seattle, Washington, 07 2022. Association for Computational Linguistics.

Minnen, G., Bond, F., and Copestake, A. Memory-Based Learning for Article Generation. In *Proceedings Of CoNLL-2000 and LLL-2000*, pages 43–48, Lisbon, Portugal, 09 2000.

Mizumoto, T. and Matsumoto, Y. Discriminative Reranking for Grammatical Error Correction with Statistical Machine Translation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1133–1138, San Diego, California, June 2016. Association for Computational Linguistics.

Naber, D. and Witt, A. *A Rule-Based Style and Grammar Checker*. GRIN Verlag, 2003. ISBN 9783640065769.

Náplava, J. and Straka, M. Grammatical Error Correction in Low-Resource Scenarios. In *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*, pages 346–356, Hong Kong, China, 11 2019. Association for Computational Linguistics.

Naseem, T. and Hussain, S. A novel approach for ranking spelling error corrections for Urdu. *Language Resources and Evaluation*, 41:117–128, 11 2007.

Olah, C. Understanding LSTM Networks. `http://colah.github.io/`, 2015. Accessed: 2022-06-01.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. 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, July 2002. Association for Computational Linguistics.

Park, J. C., Palmer, M., and Washburn, C. An English Grammar Checker as a Writing Aid for Students of English as a Second Language. In *Fifth Conference on Applied Natural Language Processing: Descriptions of System Demonstrations and Videos*, pages 24–24, Washington, DC, USA, March 1997. Association for Computational Linguistics.

Pouget-Abadie, J., Bahdanau, D., van Merriënboer, B., Cho, K., and Bengio, Y. Overcoming the Curse of Sentence Length for Neural Machine Translation using Automatic Segmentation. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 78–85, Doha, Qatar, October 2014. Association for Computational Linguistics.

Press, O. and Wolf, L. 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, April 2017. Association for Computational Linguistics.

Rosner, M., Gatt, A., Joachimsen, J., and Attard, A. Incorporating an Error Corpus into a Spellchecker for Maltese. In *8th International Conference on Language Resources and Evaluation (LREC)*. European Language Resources Association (ELRA), 5 2012.

Rozovskaya, A. and Roth, D. Algorithm Selection and Model Adaptation for ESL Correction Tasks. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 924–933, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

Samanta, P. and Chaudhuri, B. B. A simple real-word error detection and correction using local word bigram and trigram. In *Proceedings of the 25th Conference on Computational Linguistics and Speech Processing (ROCLING 2013)*, pages 211–220, Kaohsiung, Taiwan, October 2013. The Association for Computational Linguistics and Chinese Language Processing (ACLCLP).

Schwenk, H. Continuous Space Translation Models for Phrase-Based Statistical Machine Translation. In *Proceedings of COLING 2012: Posters*, pages 1071–1080, Mumbai, India, December 2012.

Sennrich, R., Haddow, B., and Birch, A. Edinburgh Neural Machine Translation Systems for WMT 16. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 371–376, Berlin, Germany, August 2016a. Association for Computational Linguistics.

Sennrich, R., Haddow, B., and Birch, A. 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, August 2016b. Association for Computational Linguistics.

Sennrich, R., Firat, O., Cho, K., Birch, A., Haddow, B., Hitschler, J., Junczys-Dowmunt, M., Läubli, S., Barone, A. V. M., Mokry, J., and Nădejde, M. Nematus: a Toolkit for Neural Machine Translation. In *Proceedings of the Software Demonstrations of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 65–68, Valencia, Spain, April 2017. Association for Computational Linguistics.

Singh, S. and Singh, S. Systematic review of spell-checkers for highly inflectional languages. *Artificial Intelligence Review*, 53:4051–4092, 8 2020.

Susanto, R. H., Phandi, P., and Ng, H. T. System Combination for Grammatical Error Correction. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 951–962, Doha, Qatar, October 2014. Association for Computational Linguistics.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to Sequence Learning with Neural Networks. *Advances in Neural Information Processing Systems*, 4, 09 2014.

Tetreault, J. R. and Chodorow, M. The Ups and Downs of Preposition Error Detection in ESL Writing. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 865–872, Manchester, UK, August 2008. Coling 2008 Organizing Committee.

Tschichold, C., Bodmer, F., Cornu, E., Grosjean, F., Grosjean, L., Kubler, N., Lewy, N., and Tschumi, C. Developing a new grammar checker for English as a second language. In *Workshop On From Research To Commercial Applications: Making NLP Work In Practice*, 01 1997.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 06 2017.

Vella, O. Spellchecking exercises for diploma students (2015). private communication, 2015.

Vella, O. Spellchecking exercises for diploma students (2022). private communication, 2022.

Wasala, A., Weerasinghe, R., Pushpananda, R., and Liyanage, C. A Data-Driven Approach to Checking and Correcting Spelling Errors in Sinhala. *International Journal on Advances in ICT for Emerging Regions (ICTer)*, pages 11–24, 12 2010.

Xie, Z., Avati, A., Arivazhagan, N., Jurafsky, D., and Ng, A. Y. Neural Language Correction with Character-Based Attention. *ArXiv*, abs/1603.09727, 03 2016.

Yuan, Z. and Briscoe, T. Grammatical error correction using neural machine translation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 380–386, San Diego, California, June 2016. Association for Computational Linguistics.

Zammit, A. A Dependency Parser for the Maltese Language using Deep Neural Networks. Master's thesis, University of Malta, 6 2018.

Additional Model Scores

In this appendix, we have included the full result sets for all the scoring metrics used for each model. The *Run* sub-column represents the score that was achieved during that batch in particular. The *Best* sub-column represents the best-accumulated score up until that batch.

Table 1: Benchmark (AMUN) - Training Scores

Updates	BLEU		Cross Entropy		CE Mean Words		Perplexity	
	Run	Best	Run	Best	Run	Best	Run	Best
1000	2.08	2.08	172.51	172.51	6.10	6.10	444.12	444.12
2000	5.40	5.40	144.72	144.72	5.11	5.11	166.32	166.32
3000	8.14	8.14	121.88	121.88	4.31	4.31	99.47	99.47
4000	23.21	23.21	86.42	86.42	3.13	3.13	22.81	22.81
5000	19.04	23.21	94.49	86.42	3.42	3.13	30.52	22.81
6000	35.77	35.77	75.21	74.87	2.73	2.71	15.20	15.02
7000	35.79	37.22	71.53	71.53	2.59	2.59	13.26	13.26
8000	47.33	47.33	62.62	62.62	2.27	2.27	9.69	9.69
9000	50.60	50.60	46.74	46.74	1.66	1.66	5.28	5.28
10000	59.54	60.62	40.59	40.03	1.54	1.51	4.65	4.56
11000	62.43	62.43	41.98	40.03	1.57	1.51	4.91	4.56
12000	61.34	63.95	40.53	39.09	1.54	1.48	4.65	4.41
13000	70.90	70.90	32.24	32.24	1.16	1.16	3.20	3.20
14000	68.38	72.41	34.59	32.24	1.25	1.16	3.47	3.20
15000	77.32	77.32	27.43	27.43	0.98	0.98	2.68	2.68
16000	71.34	77.32	29.83	27.43	1.07	0.98	2.92	2.68
17000	76.71	77.32	24.43	24.43	0.93	0.93	2.53	2.53
18000	75.33	77.32	26.58	24.43	1.01	0.93	2.75	2.53
19000	76.75	77.32	25.85	24.43	0.92	0.92	2.50	2.50
20000	77.49	78.12	25.65	24.43	0.90	0.89	2.46	2.44

Table 2: Benchmark (Seq2Seq) - Training Scores

Updates	BLEU		Cross Entropy		CE Mean Words		Perplexity	
	Run	Best	Run	Best	Run	Best	Run	Best
1000	1.59	1.64	178.80	178.80	6.29	6.29	540.67	540.67
2000	3.90	3.90	157.10	157.10	5.53	5.53	254.26	254.26
3000	10.59	10.59	115.80	115.80	4.12	4.12	61.78	61.78
4000	19.42	19.42	90.88	90.88	3.25	3.25	25.76	25.76
5000	24.13	24.13	82.56	82.56	2.95	2.95	19.16	19.16
6000	25.84	28.85	77.04	77.04	2.76	2.76	15.73	15.73
7000	43.76	43.76	53.94	53.94	1.98	1.98	7.23	7.23
8000	50.55	50.55	51.72	51.72	1.90	1.90	6.74	6.74
9000	51.95	52.57	51.05	50.41	1.88	1.85	6.52	6.35
10000	54.46	56.29	49.26	48.97	1.81	1.80	6.14	6.03
11000	65.77	65.77	37.71	37.71	1.33	1.33	3.77	3.77
12000	65.78	66.81	38.47	37.68	1.35	1.33	3.87	3.77
13000	70.50	70.50	33.71	33.71	1.20	1.20	3.33	3.33
14000	69.78	71.71	35.54	33.62	1.27	1.20	3.55	3.32
15000	76.82	76.82	28.07	28.07	1.03	1.03	2.81	2.81
16000	76.75	77.08	28.13	27.89	1.03	1.03	2.81	2.79
17000	74.19	77.08	29.44	27.89	1.08	1.03	2.95	2.79
18000	72.31	77.08	31.26	27.89	1.15	1.03	3.16	2.79
19000	71.65	77.08	32.99	27.89	1.21	1.03	3.37	2.79
20000	77.28	78.37	27.70	27.70	0.97	0.97	2.64	2.64

Table 3: Benchmark (Vaswani Transformer) - Training Scores

Updates	BLEU		Cross Entropy		CE Mean Words		Perplexity	
	Run	Best	Run	Best	Run	Best	Run	Best
1000	1.39	1.39	165.68	165.68	5.98	5.98	393.71	393.71
2000	4.51	4.51	140.50	140.50	5.07	5.07	158.80	158.80
3000	11.78	11.78	119.25	119.25	4.30	4.30	73.78	73.78
4000	20.72	20.72	103.06	103.06	3.72	3.72	41.14	41.14
5000	27.58	27.58	91.90	91.90	3.31	3.31	27.51	27.51
6000	34.58	34.58	84.10	84.10	3.03	3.03	20.77	20.77
7000	38.53	38.53	78.49	78.49	2.83	2.83	16.96	16.96
8000	43.39	43.39	74.42	74.42	2.68	2.68	14.64	14.64
9000	63.54	63.54	37.87	37.87	1.33	1.33	3.78	3.78
10000	63.23	63.54	38.10	37.70	1.34	1.32	3.81	3.76
11000	63.01	63.54	39.33	37.70	1.38	1.32	3.98	3.76
12000	62.94	63.54	40.60	37.70	1.43	1.32	4.16	3.76
13000	62.47	63.54	41.58	37.70	1.46	1.32	4.31	3.76
14000	69.78	75.50	28.61	21.98	1.04	0.79	2.80	2.20
15000	68.57	75.50	30.37	21.98	1.09	0.79	2.98	2.20
16000	67.47	75.50	33.08	21.98	1.19	0.79	3.29	2.20
17000	63.27	75.50	36.21	21.98	1.30	0.79	3.68	2.20
18000	67.85	75.50	36.43	21.98	1.31	0.79	3.71	2.20
19000	61.24	75.50	37.52	21.98	1.35	0.79	3.85	2.20
20000	77.80	77.80	17.45	17.45	0.61	0.61	1.85	1.85

Table 4: Source Word Corruption - Training Scores

Updates	BLEU		Cross Entropy		CE Mean Words		Perplexity	
	Run	Best	Run	Best	Run	Best	Run	Best
1000	0.92	0.92	168.74	168.74	6.28	6.28	534.65	534.65
2000	2.20	2.20	154.84	154.84	5.73	5.73	307.88	307.88
3000	6.93	6.93	132.42	132.42	4.90	4.90	134.19	134.19
4000	9.11	10.40	118.02	118.02	4.37	4.37	78.75	78.75
5000	19.75	19.75	106.10	106.10	3.93	3.93	50.68	50.68
6000	21.45	25.53	80.16	80.16	2.90	2.90	18.11	18.11
7000	31.39	31.39	75.57	75.57	2.73	2.73	15.34	15.34
8000	36.03	36.03	71.61	71.61	2.59	2.59	13.30	13.30
9000	39.73	39.73	67.97	67.97	2.46	2.46	11.66	11.66
10000	44.03	44.03	64.80	64.80	2.34	2.34	10.40	10.40
11000	55.02	55.02	62.16	61.74	2.25	2.23	9.45	9.31
12000	53.30	58.55	43.43	43.43	1.59	1.59	4.92	4.92
13000	56.72	58.55	43.12	43.12	1.58	1.58	4.86	4.86
14000	58.76	59.16	43.25	43.12	1.59	1.58	4.89	4.86
15000	58.45	59.16	43.47	43.12	1.59	1.58	4.93	4.86
16000	60.46	60.46	43.73	43.12	1.60	1.58	4.97	4.86
17000	57.71	60.50	43.89	43.12	1.61	1.58	5.00	4.86
18000	60.84	60.84	44.05	43.12	1.62	1.58	5.03	4.86
19000	68.66	68.66	44.11	43.12	1.62	1.58	5.04	4.86
20000	64.54	69.26	30.70	29.85	1.12	1.09	3.08	2.98

Table 5: Benchmark + Tied Embeddings - Training Scores

Updates	BLEU		Cross Entropy		CE Mean Words		Perplexity	
	Run	Best	Run	Best	Run	Best	Run	Best
1000	1.24	1.24	167.37	167.37	6.20	6.20	494.28	494.28
2000	3.58	3.58	148.13	148.13	5.49	5.49	242.23	242.23
3000	7.46	7.46	129.19	129.19	4.79	4.79	120.08	120.08
4000	12.31	12.31	111.91	111.91	4.15	4.15	63.29	63.29
5000	23.51	23.51	74.77	74.77	2.78	2.78	16.20	16.20
6000	30.14	30.14	66.41	66.41	2.47	2.47	11.86	11.86
7000	38.14	38.14	62.25	62.25	2.32	2.32	10.16	10.16
8000	45.44	45.44	58.53	58.53	2.18	2.18	8.84	8.84
9000	48.96	48.96	55.60	55.60	2.07	2.07	7.93	7.93
10000	51.23	51.84	53.11	53.11	1.98	1.98	7.23	7.23
11000	71.33	71.33	32.81	32.81	1.21	1.21	3.34	3.34
12000	71.03	71.61	32.45	32.45	1.19	1.19	3.30	3.30
13000	69.25	71.61	33.12	32.45	1.22	1.19	3.38	3.30
14000	66.84	71.61	33.28	32.45	1.22	1.19	3.40	3.30
15000	68.16	71.61	33.51	32.45	1.23	1.19	3.43	3.30
16000	78.13	78.13	25.26	25.26	0.89	0.89	2.43	2.43
17000	74.90	78.47	25.23	25.07	0.89	0.88	2.42	2.41
18000	75.07	78.47	26.04	25.07	0.91	0.88	2.49	2.41
19000	75.39	78.47	27.03	25.07	0.95	0.88	2.58	2.41
20000	79.26	79.26	19.62	19.62	0.71	0.71	2.03	2.03

Table 6: Benchmark + Pretrained (BERTu) - Training Scores

Updates	BLEU		Cross Entropy		CE Mean Words		Perplexity	
	Run	Best	Run	Best	Run	Best	Run	Best
1000	2.90	2.90	161.79	161.79	5.66	5.66	286.71	286.71
2000	6.91	6.91	126.55	126.55	4.43	4.43	83.61	83.61
3000	23.39	23.39	84.66	84.66	2.98	2.98	19.77	19.77
4000	36.04	36.04	72.24	72.24	2.55	2.55	12.76	12.76
5000	46.02	46.02	63.66	63.66	2.24	2.24	9.43	9.43
6000	48.07	48.07	57.94	57.94	2.04	2.04	7.71	7.71
7000	52.94	52.94	53.95	53.95	1.90	1.90	6.70	6.70
8000	72.12	72.12	32.27	32.27	1.19	1.19	3.30	3.30
9000	71.85	72.12	32.50	32.20	1.20	1.19	3.32	3.29
10000	71.65	72.12	32.87	32.20	1.22	1.19	3.37	3.29
11000	71.08	72.12	33.15	32.20	1.23	1.19	3.41	3.29
12000	70.84	72.12	33.45	32.20	1.24	1.19	3.44	3.29
13000	79.90	79.90	21.55	21.55	0.77	0.77	2.16	2.16
14000	79.64	80.11	22.10	21.55	0.79	0.77	2.20	2.16
15000	77.63	80.11	23.67	21.55	0.85	0.77	2.33	2.16
16000	75.67	80.11	25.03	21.55	0.90	0.77	2.45	2.16
17000	74.53	80.11	26.13	21.55	0.93	0.77	2.55	2.16
18000	83.86	83.86	17.26	17.26	0.63	0.63	1.87	1.87
19000	83.91	83.91	17.00	16.98	0.62	0.62	1.85	1.85
20000	83.10	83.91	17.86	16.98	0.65	0.62	1.91	1.85

Table 7: Benchmark + Pretrained (mBERTu) - Training Scores

Updates	BLEU		Cross Entropy		CE Mean Words		Perplexity	
	Run	Best	Run	Best	Run	Best	Run	Best
1000	2.90	2.90	150.57	150.57	5.43	5.43	254.13	254.13
2000	6.90	6.90	112.77	112.77	4.07	4.07	72.07	72.07
3000	23.39	23.39	71.98	71.98	2.60	2.60	17.09	17.09
4000	39.04	39.04	66.15	66.15	2.39	2.39	11.00	11.00
5000	57.56	57.56	50.53	50.53	1.82	1.82	6.19	6.19
6000	60.73	60.73	47.53	47.53	1.72	1.72	5.56	5.56
7000	61.99	61.99	45.42	45.42	1.64	1.64	5.15	5.15
8000	63.13	63.13	43.89	43.89	1.58	1.58	4.87	4.87
9000	75.98	75.98	27.20	27.20	1.00	1.00	2.72	2.72
10000	76.37	76.37	26.57	26.50	0.98	0.97	2.65	2.65
11000	75.77	76.37	27.62	26.50	1.01	0.97	2.76	2.65
12000	74.80	76.37	28.62	26.50	1.05	0.97	2.86	2.65
13000	74.05	76.37	29.27	26.50	1.07	0.97	2.93	2.65
14000	82.70	82.70	17.97	17.97	0.64	0.64	1.89	1.89
15000	83.23	83.49	17.61	17.58	0.63	0.62	1.87	1.87
16000	81.75	83.49	18.80	17.58	0.67	0.62	1.95	1.87
17000	80.54	83.49	20.49	17.58	0.73	0.62	2.07	1.87
18000	78.75	83.49	21.94	17.58	0.78	0.62	2.18	1.87
19000	87.41	87.41	14.29	14.29	0.50	0.50	1.65	1.65
20000	86.72	87.41	14.68	14.29	0.52	0.50	1.68	1.65

Table 8: Domain & Error Adapted - Training Scores

Updates	BLEU		Cross Entropy		CE Mean Words		Perplexity	
	Run	Best	Run	Best	Run	Best	Run	Best
1000	0.37	0.37	120.60	120.60	6.05	6.05	426.07	426.07
2000	2.01	2.04	105.63	105.63	5.29	5.29	200.03	200.03
3000	7.45	7.45	90.88	90.88	4.55	4.55	95.94	95.94
4000	12.65	12.65	72.95	72.95	3.81	3.81	45.57	45.57
5000	16.28	16.28	66.53	66.53	3.51	3.51	32.85	32.85
6000	21.17	21.83	60.27	60.27	3.16	3.16	23.50	23.50
7000	25.37	25.37	52.75	52.75	2.76	2.76	15.75	15.75
8000	32.72	32.72	48.56	48.56	2.54	2.54	12.66	12.66
9000	29.40	32.72	46.51	46.51	2.43	2.43	11.22	11.22
10000	34.95	35.95	42.85	42.85	2.24	2.24	9.40	9.40
11000	40.78	40.78	41.11	41.11	2.15	2.15	8.60	8.60
12000	44.82	44.82	37.12	36.66	1.91	1.89	6.75	6.57
13000	46.42	46.42	36.35	36.12	1.87	1.86	6.47	6.41
14000	44.85	46.42	35.19	35.19	1.81	1.81	6.10	6.10
15000	51.67	51.67	34.74	34.74	1.79	1.79	5.95	5.95
16000	50.58	51.99	34.02	34.02	1.75	1.75	5.75	5.75
17000	51.00	51.99	34.42	34.02	1.78	1.75	5.90	5.75
18000	56.65	56.65	30.23	30.23	1.51	1.51	4.52	4.52
19000	53.51	56.83	29.82	29.66	1.49	1.48	4.44	4.41
20000	55.41	56.83	29.69	29.66	1.48	1.48	4.41	4.41

Table 9: Domain & Error Adapted + Large Vocabularies - Training Scores

Updates	BLEU		Cross Entropy		CE Mean Words		Perplexity	
	Run	Best	Run	Best	Run	Best	Run	Best
1000	0.61	0.61	127.56	127.56	6.46	6.46	639.38	639.38
2000	1.09	1.09	116.73	116.73	5.91	5.91	369.43	369.43
3000	3.57	3.57	100.71	100.71	5.27	5.27	194.38	194.38
4000	6.74	6.74	91.34	91.34	4.78	4.78	119.00	119.00
5000	10.79	10.79	82.42	82.42	4.31	4.31	74.65	74.65
6000	15.61	15.61	74.36	74.36	3.89	3.89	48.96	48.96
7000	20.68	20.68	67.41	67.41	3.53	3.53	34.04	34.04
8000	26.20	26.20	61.58	61.58	3.22	3.22	25.09	25.09
9000	30.46	30.46	56.04	56.04	2.77	2.77	15.91	15.91
10000	33.27	33.27	53.43	53.43	2.64	2.64	13.99	13.99
11000	40.90	40.90	47.82	47.82	2.41	2.41	11.10	11.10
12000	41.60	41.60	46.18	46.18	2.32	2.32	10.22	10.22
13000	43.43	43.95	43.56	43.56	2.18	2.18	8.87	8.87
14000	45.23	45.86	43.31	43.31	2.17	2.17	8.76	8.76
15000	46.69	46.69	42.86	42.86	2.15	2.15	8.57	8.57
16000	47.90	47.90	42.29	42.29	2.12	2.12	8.33	8.33
17000	50.17	50.17	41.83	41.83	2.10	2.10	8.13	8.13
18000	51.54	51.54	39.44	39.44	1.99	1.99	7.28	7.28
19000	51.79	51.79	39.44	39.44	1.99	1.99	7.28	7.28
20000	52.51	52.51	39.54	39.41	1.95	1.95	7.06	7.06

Table 10: Domain & Error Adapted + Tied Embeddings - Training Scores

Updates	BLEU		Cross Entropy		CE Mean Words		Perplexity	
	Run	Best	Run	Best	Run	Best	Run	Best
1000	0.53	0.53	126.51	126.51	6.37	6.37	584.27	584.27
2000	2.36	2.36	110.67	110.67	5.57	5.57	263.24	263.24
3000	4.02	4.02	95.46	95.46	4.72	4.72	111.97	111.97
4000	7.96	7.96	82.22	82.22	4.06	4.06	58.19	58.19
5000	14.47	14.47	70.51	70.51	3.48	3.48	32.62	32.62
6000	21.37	21.37	62.58	62.58	3.09	3.09	22.05	22.05
7000	30.08	30.08	54.23	54.23	2.71	2.71	15.05	15.05
8000	37.52	37.52	51.25	51.25	2.56	2.56	12.96	12.96
9000	41.50	41.50	48.60	48.60	2.43	2.43	11.35	11.35
10000	44.60	44.60	46.63	46.63	2.33	2.33	10.29	10.29
11000	48.09	48.09	41.48	41.48	2.09	2.09	8.06	8.06
12000	48.65	48.65	42.34	40.80	2.04	2.04	7.67	7.67
13000	50.41	50.41	41.23	40.80	1.98	1.98	7.27	7.27
14000	52.16	52.16	40.54	40.54	1.95	1.95	7.03	7.03
15000	52.19	52.19	39.82	39.82	1.92	1.92	6.79	6.79
16000	57.49	57.49	34.61	34.61	1.73	1.73	5.66	5.66
17000	57.59	58.03	34.12	34.12	1.71	1.71	5.52	5.52
18000	58.45	58.45	33.87	33.87	1.70	1.70	5.45	5.45
19000	57.88	58.99	33.72	33.72	1.69	1.69	5.41	5.41
20000	59.36	59.36	33.65	33.65	1.69	1.69	5.39	5.39

Table 11: Domain & Error Adapted + Pretrained (BERTu) - Training Scores

Updates	BLEU		Cross Entropy		CE Mean Words		Perplexity	
	Run	Best	Run	Best	Run	Best	Run	Best
1000	2.98	2.98	115.55	115.55	5.64	5.64	280.58	280.58
2000	7.83	7.83	91.43	91.43	4.46	4.46	86.49	86.49
3000	16.22	16.22	71.28	71.28	3.57	3.57	35.43	35.43
4000	28.37	28.37	59.25	59.25	2.94	2.94	18.96	18.96
5000	37.12	37.12	53.38	53.38	2.65	2.65	14.17	14.17
6000	42.54	42.54	49.26	49.26	2.45	2.45	11.55	11.55
7000	48.99	48.99	43.08	43.08	2.14	2.14	8.48	8.48
8000	52.07	52.07	40.88	40.88	2.03	2.03	7.60	7.60
9000	54.01	54.01	39.03	39.03	1.94	1.94	6.93	6.93
10000	56.13	56.13	37.81	37.81	1.88	1.88	6.53	6.53
11000	57.27	57.27	36.85	36.85	1.83	1.83	6.23	6.23
12000	60.48	60.48	33.76	33.76	1.65	1.65	5.21	5.21
13000	59.82	60.48	33.43	33.43	1.63	1.63	5.13	5.13
14000	64.28	64.28	29.32	29.32	1.48	1.48	4.38	4.38
15000	65.02	65.02	28.93	28.93	1.46	1.46	4.30	4.30
16000	63.65	65.02	28.87	28.87	1.46	1.46	4.29	4.29
17000	65.67	65.67	27.98	27.98	1.41	1.41	4.08	4.08
18000	65.16	65.67	27.96	27.95	1.41	1.41	4.08	4.08
19000	65.47	65.69	28.07	27.95	1.41	1.41	4.10	4.08
20000	69.15	69.15	24.31	24.31	1.25	1.25	3.50	3.50

Table 12: Domain & Error Adapted + Pretrained (mBERTu) - Training Scores

Updates	BLEU		Cross Entropy		CE Mean Words		Perplexity	
	Run	Best	Run	Best	Run	Best	Run	Best
1000	3.57	3.57	115.77	115.77	5.74	5.74	312.28	312.28
2000	7.62	7.62	87.69	87.69	4.52	4.52	91.64	91.64
3000	14.18	14.18	78.91	78.91	4.07	4.07	58.30	58.30
4000	19.09	19.09	72.17	72.17	3.72	3.72	41.20	41.20
5000	28.98	28.98	59.68	59.68	2.98	2.98	19.64	19.64
6000	38.26	38.26	50.69	50.69	2.55	2.55	12.85	12.85
7000	42.29	42.29	48.41	48.41	2.44	2.44	11.45	11.45
8000	45.19	45.19	48.03	48.03	2.42	2.42	11.24	11.24
9000	50.41	50.41	42.20	42.20	2.12	2.12	8.37	8.37
10000	50.54	50.54	42.42	42.06	2.14	2.12	8.47	8.32
11000	49.19	50.54	42.82	42.06	2.16	2.12	8.64	8.32
12000	57.73	57.73	37.64	37.64	1.87	1.87	6.48	6.48
13000	56.80	57.73	37.92	37.52	1.88	1.86	6.57	6.44
14000	55.97	57.73	38.83	37.52	1.93	1.86	6.87	6.44
15000	56.76	57.73	39.37	37.52	1.95	1.86	7.06	6.44
16000	54.47	57.73	39.84	37.52	1.98	1.86	7.22	6.44
17000	54.27	57.73	40.26	37.52	2.00	1.86	7.38	6.44
18000	60.16	60.16	33.71	33.71	1.69	1.69	5.43	5.43
19000	60.81	60.81	33.52	33.42	1.68	1.68	5.38	5.35
20000	59.77	60.81	34.33	33.42	1.72	1.68	5.60	5.35

Table 13: Final Model - Training Scores

Updates	BLEU		Cross Entropy		CE Mean Words		Perplexity	
	Run	Best	Run	Best	Run	Best	Run	Best
1000	0.97	0.97	134.16	134.16	6.32	6.32	240.15	240.15
2000	4.20	4.20	125.07	125.07	5.89	5.89	166.18	166.18
3000	6.42	6.42	94.06	94.06	4.43	4.43	84.15	84.15
4000	14.35	14.35	78.33	78.33	3.61	3.61	36.85	36.85
5000	22.09	22.09	70.83	70.83	3.26	3.26	26.09	26.09
6000	28.16	28.16	60.58	60.58	2.86	2.86	17.50	17.50
7000	33.83	33.83	55.61	55.61	2.63	2.63	13.89	13.89
8000	37.03	37.03	52.73	52.73	2.50	2.50	12.13	12.13
9000	40.51	40.51	52.32	52.32	2.48	2.48	11.89	11.89
10000	42.64	42.64	51.56	51.56	2.44	2.44	11.47	11.47
11000	45.44	45.98	44.20	44.20	2.13	2.13	8.44	8.44
12000	46.24	46.24	43.85	43.74	2.12	2.11	8.30	8.26
13000	50.92	50.92	42.09	42.09	2.01	2.01	7.47	7.47
14000	51.08	51.08	41.99	41.79	2.01	2.00	7.43	7.36
15000	53.38	53.47	39.57	39.57	1.86	1.86	6.43	6.43
16000	53.71	53.95	39.16	39.11	1.84	1.84	6.31	6.30
17000	54.97	54.97	37.42	37.42	1.77	1.77	5.87	5.87
18000	55.04	55.44	37.40	37.26	1.77	1.76	5.86	5.82
19000	53.97	55.44	38.17	37.26	1.81	1.76	6.08	5.82
20000	54.48	55.44	38.96	37.26	1.84	1.76	6.31	5.82

Plots

In this appendix, we have included the visualisations for all the scoring metrics used for each model. In the case of perplexity, we purposely eliminated the upper-bounds of the chart to focus in on the part of the curve that shows the differences in the later epochs (5000 and beyond).

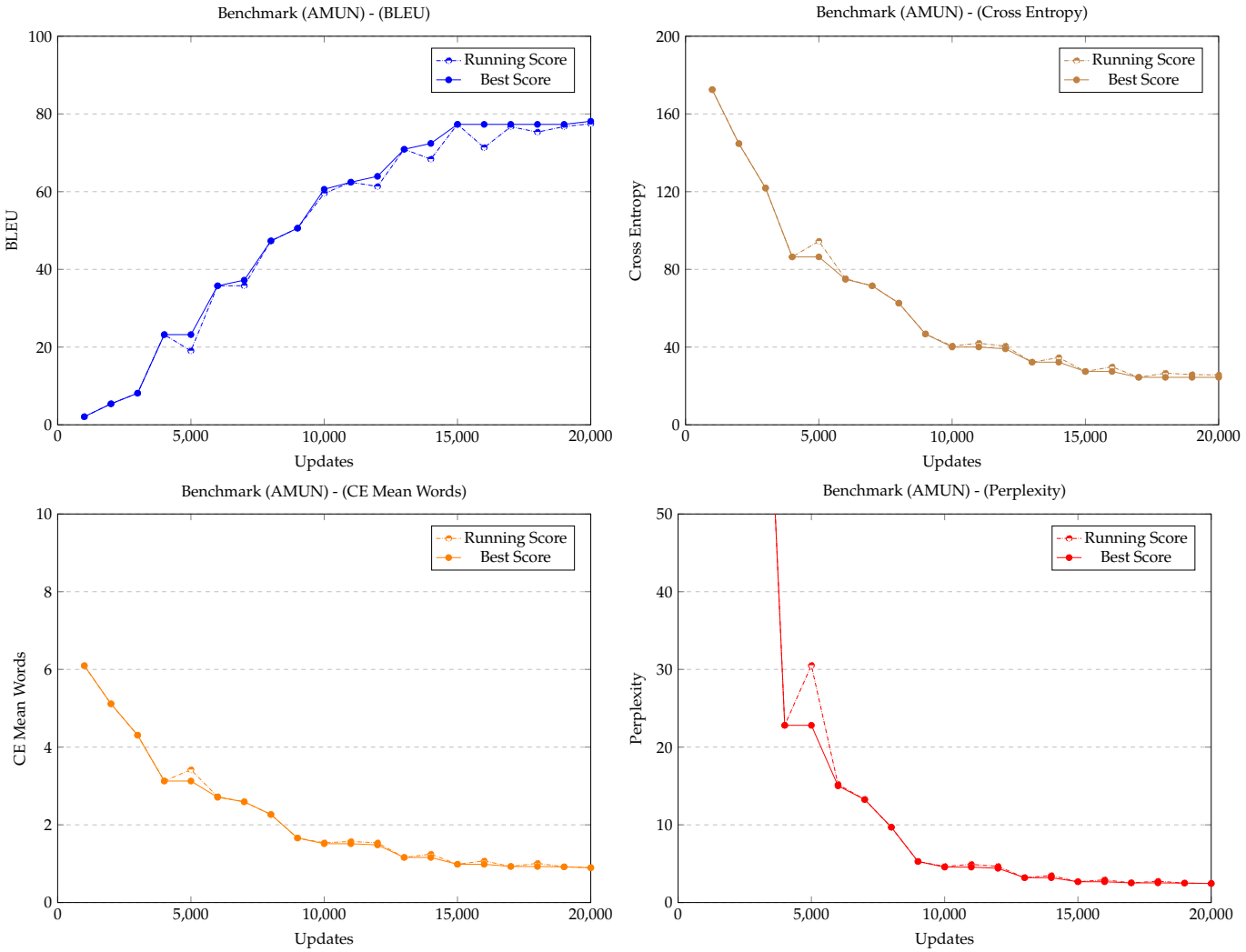

Figure 1: Benchmark (AMUN)

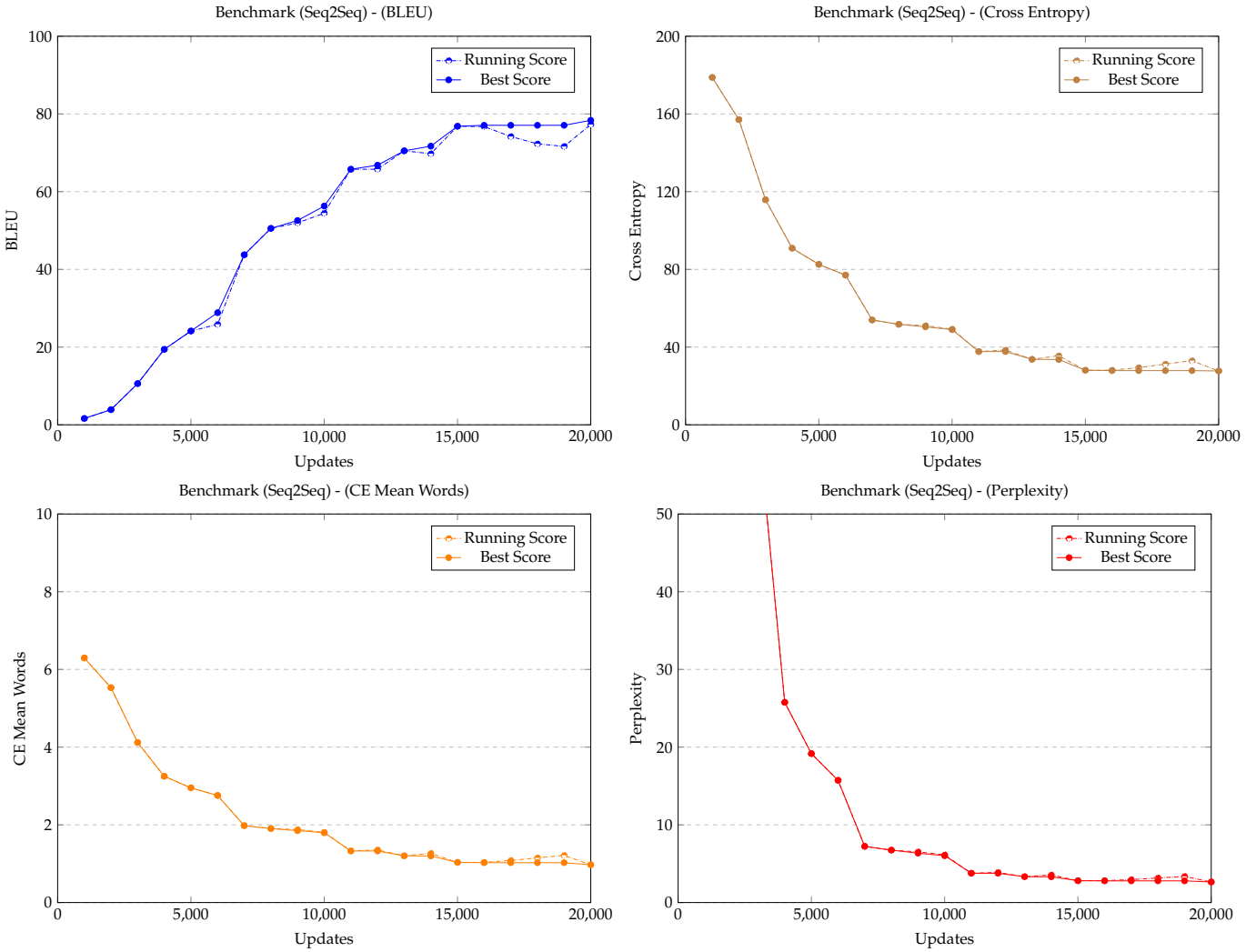

Figure 2: Benchmark (Seq2Seq)

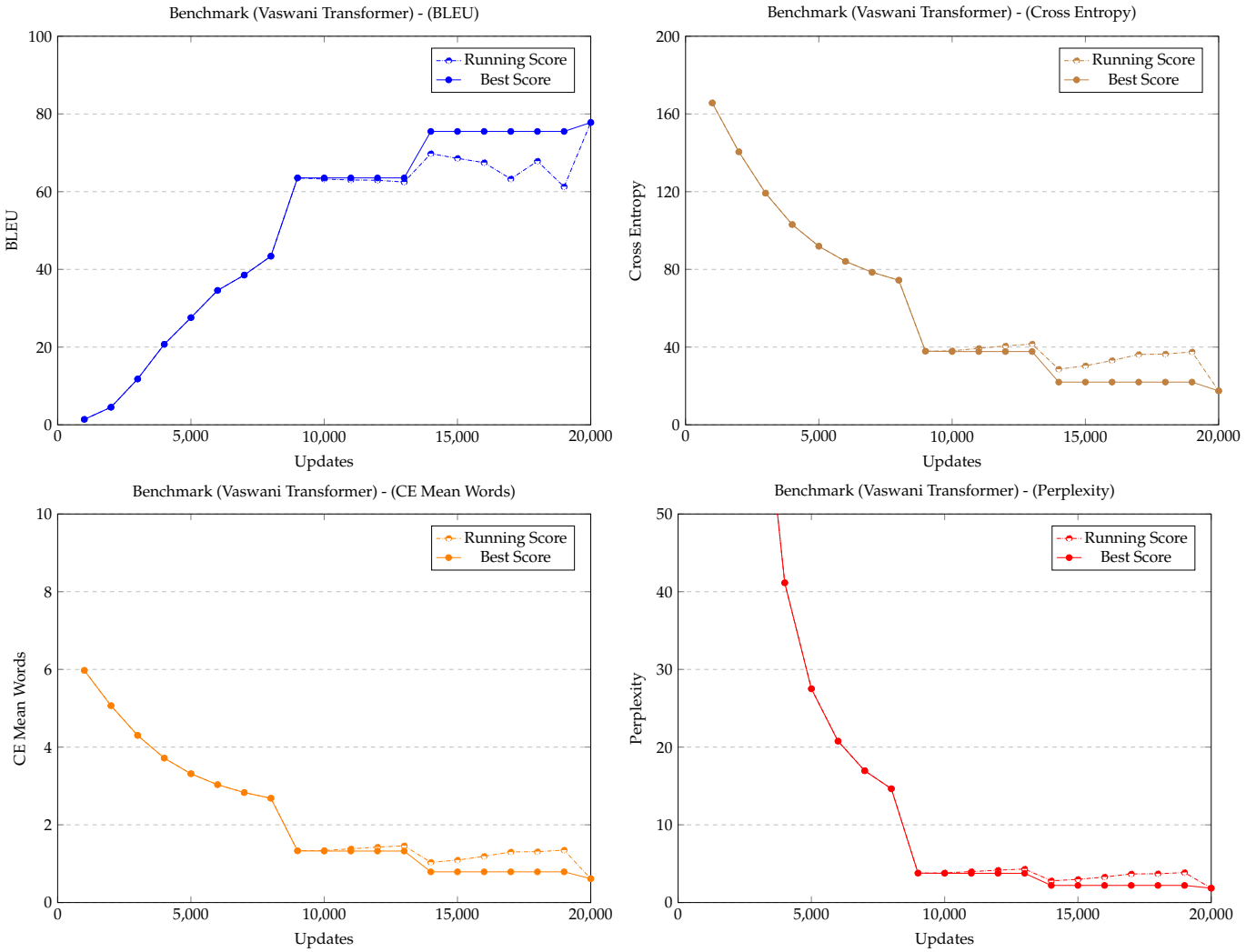

Figure 3: Benchmark (Vaswani Transformer)

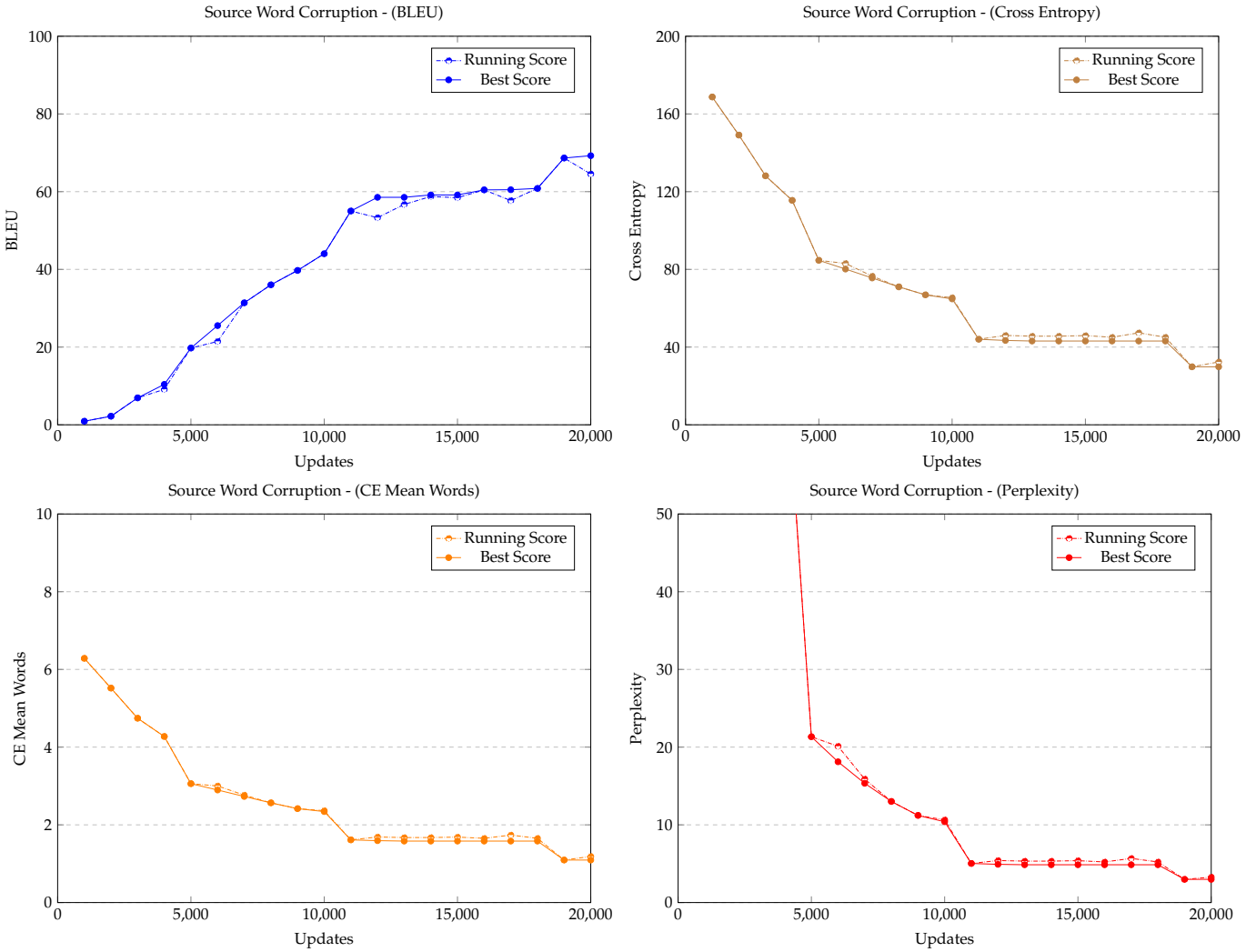

Figure 4: Source Word Corruption

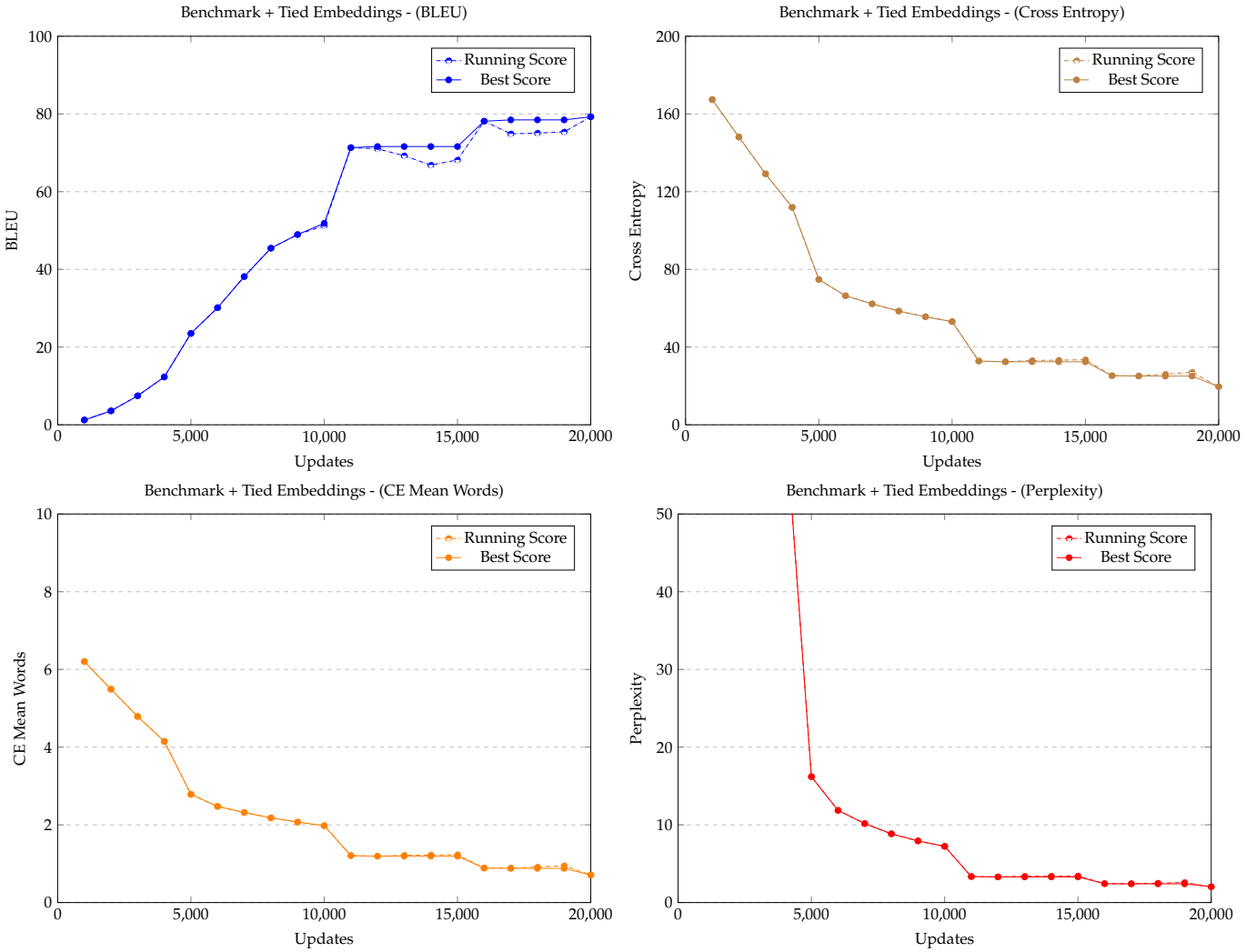

Figure 5: Benchmark + Tied Embeddings

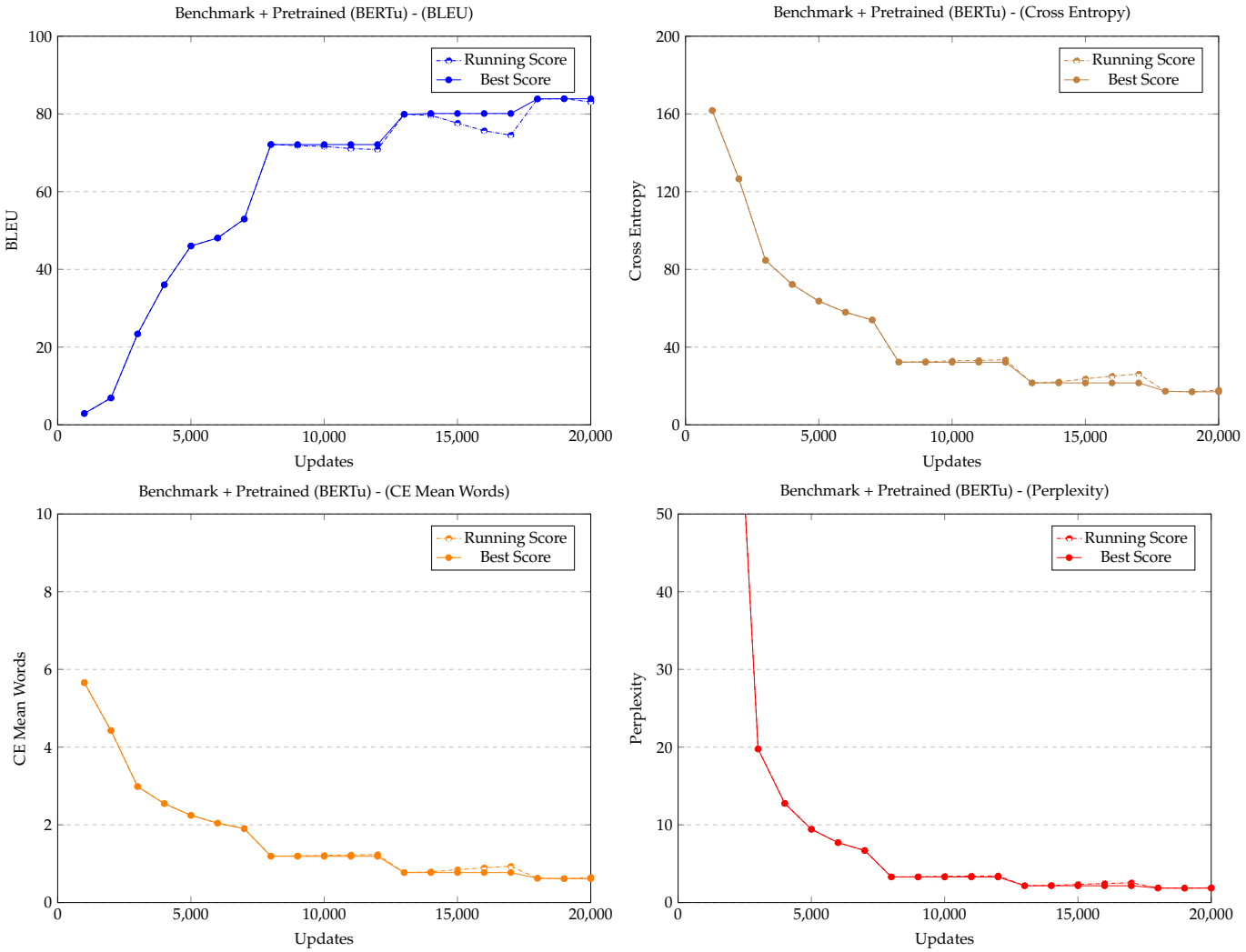

Figure 6: Benchmark + Pretrained (BERTu)

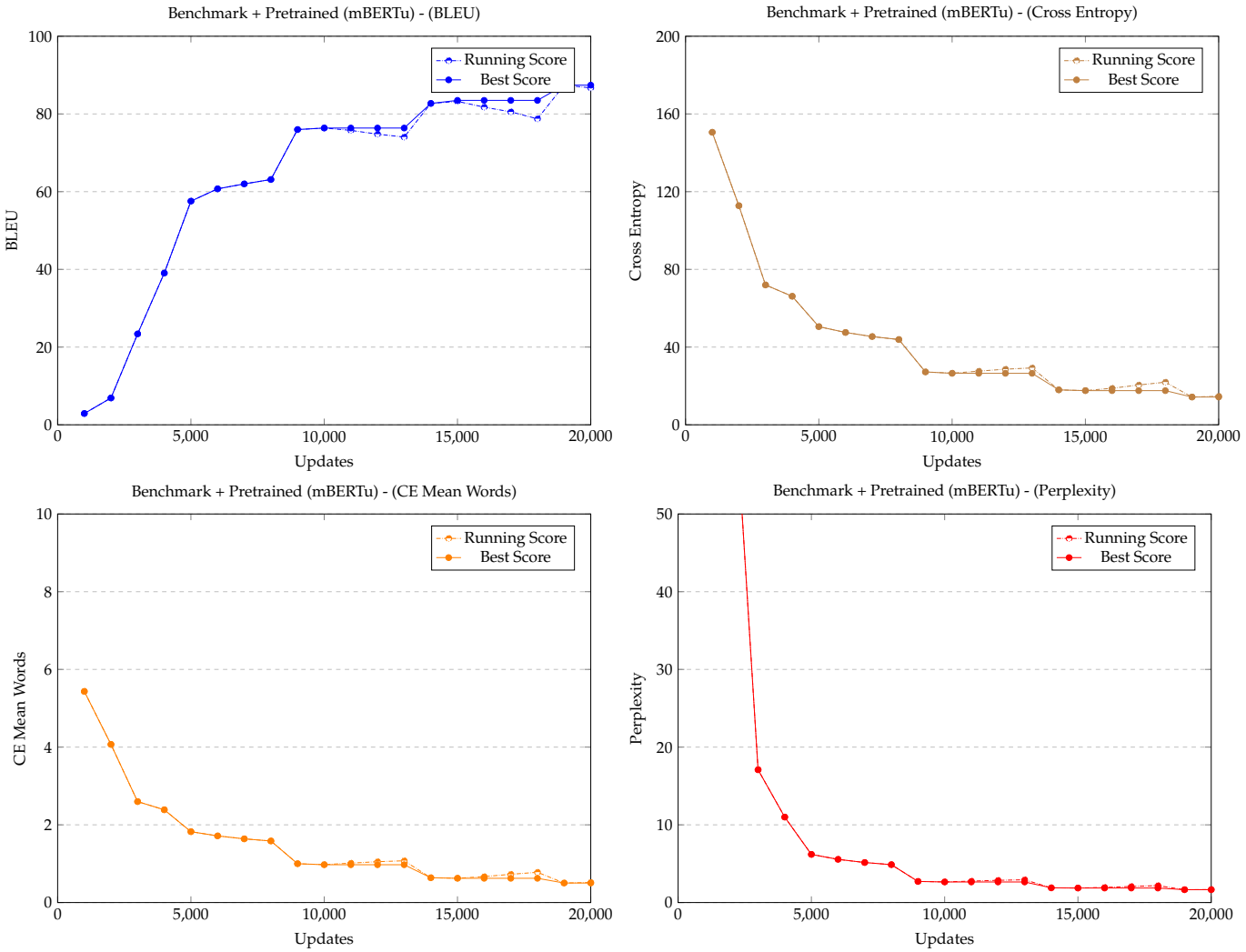

Figure 7: Benchmark + Pretrained (mBERTu)

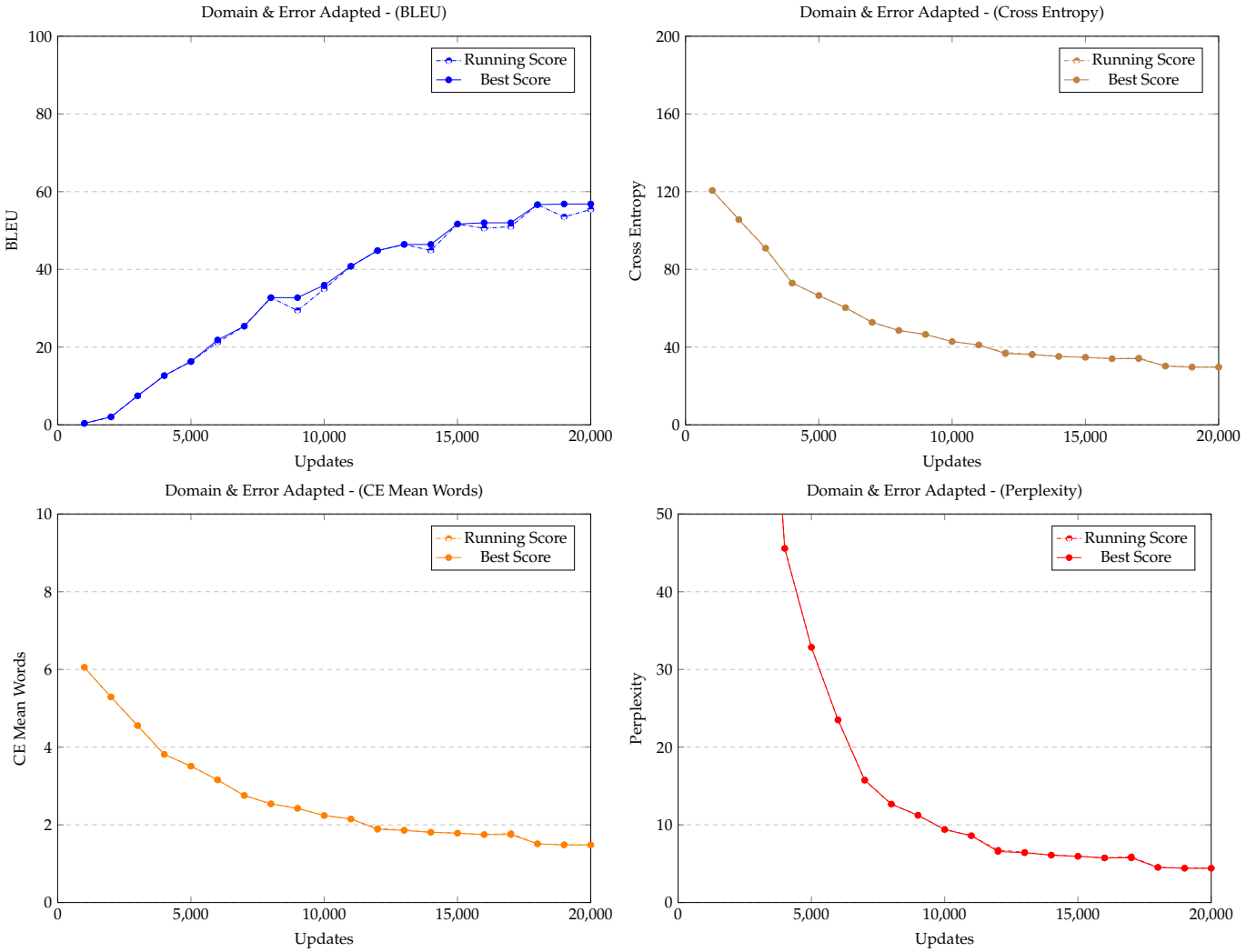

Figure 8: Domain & Error Adapted

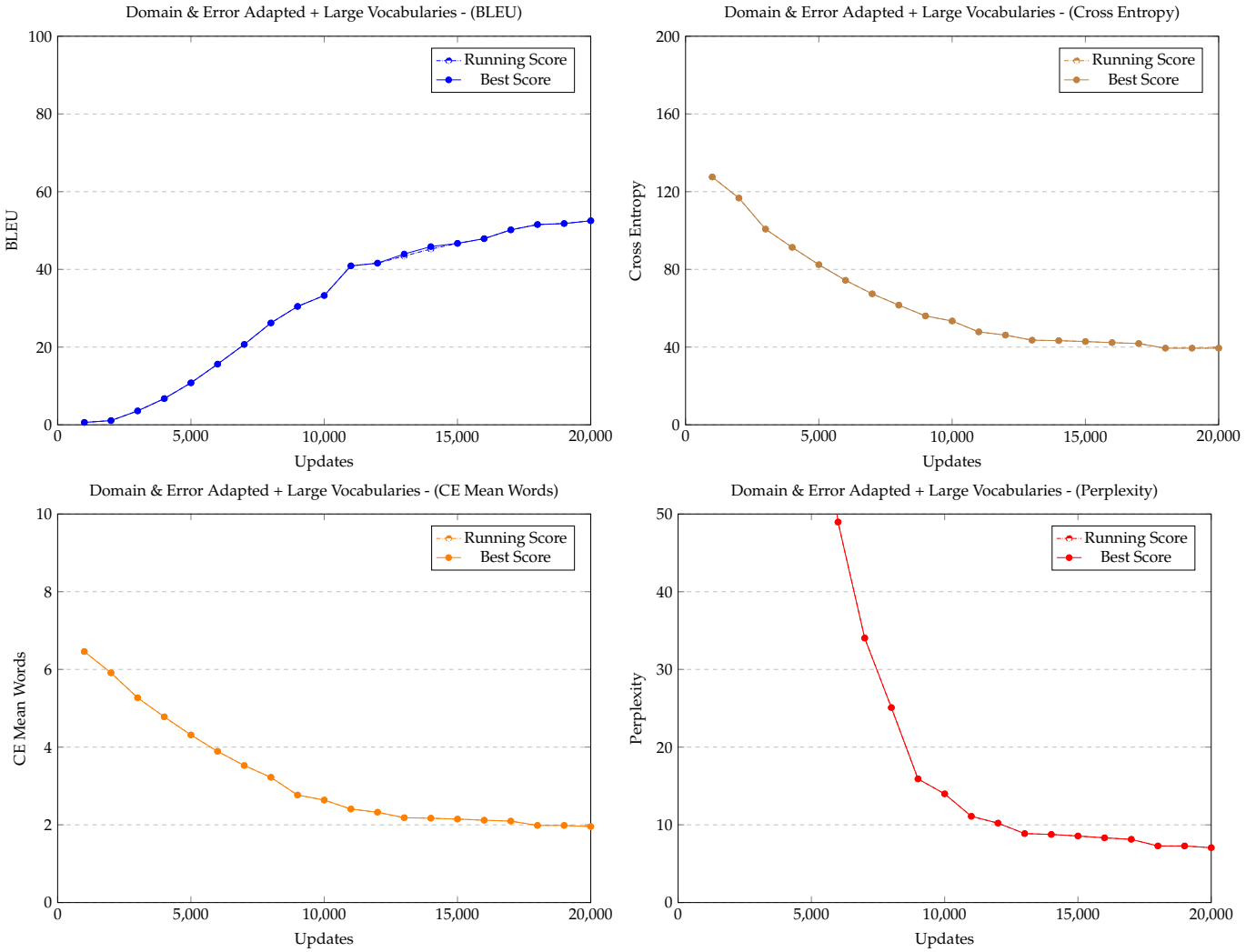

Figure 9: Domain & Error Adapted + Large Vocabularies

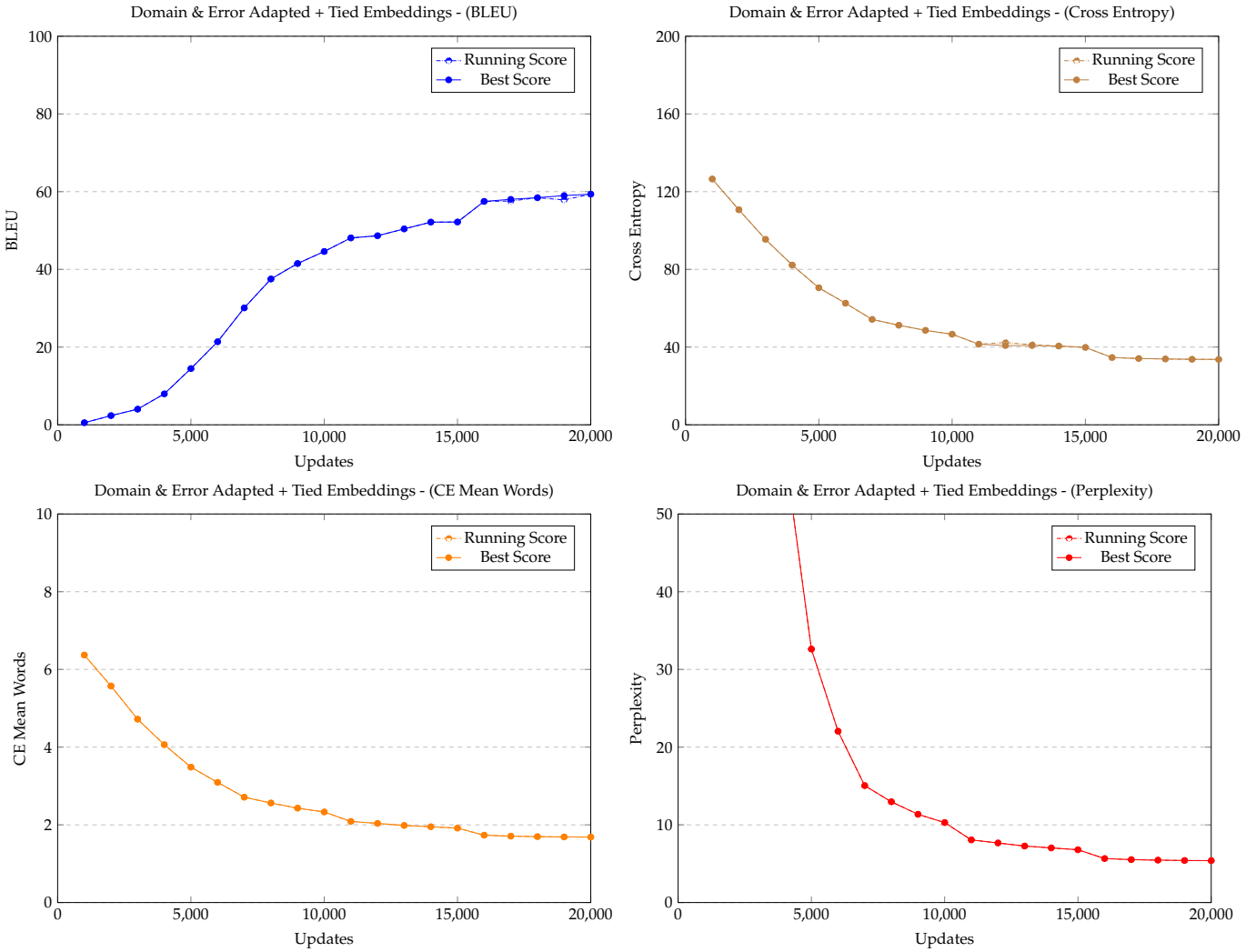

Figure 10: Domain & Error Adapted + Tied Embeddings

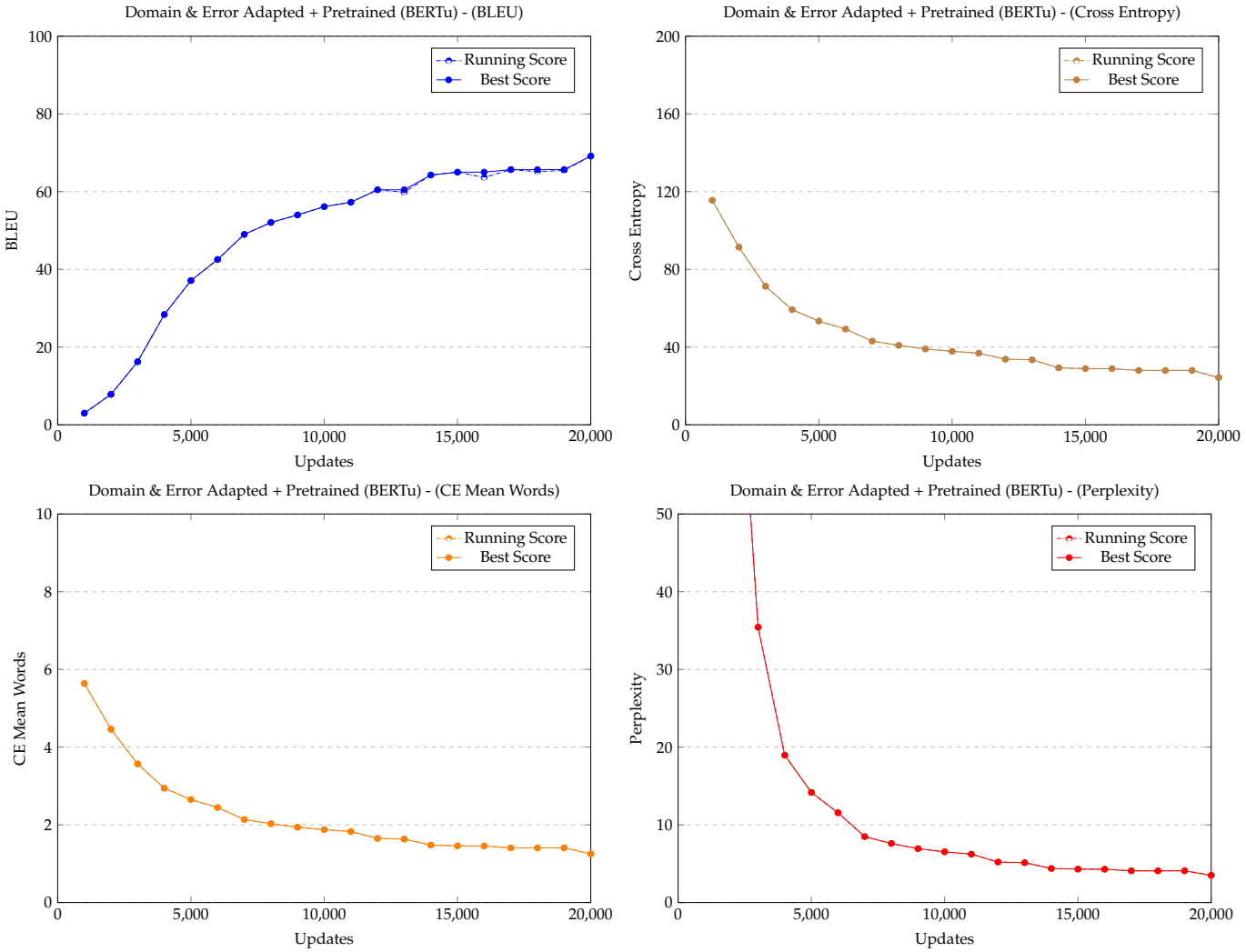

Figure 11: Domain & Error Adapted + Pretrained (BERTu)

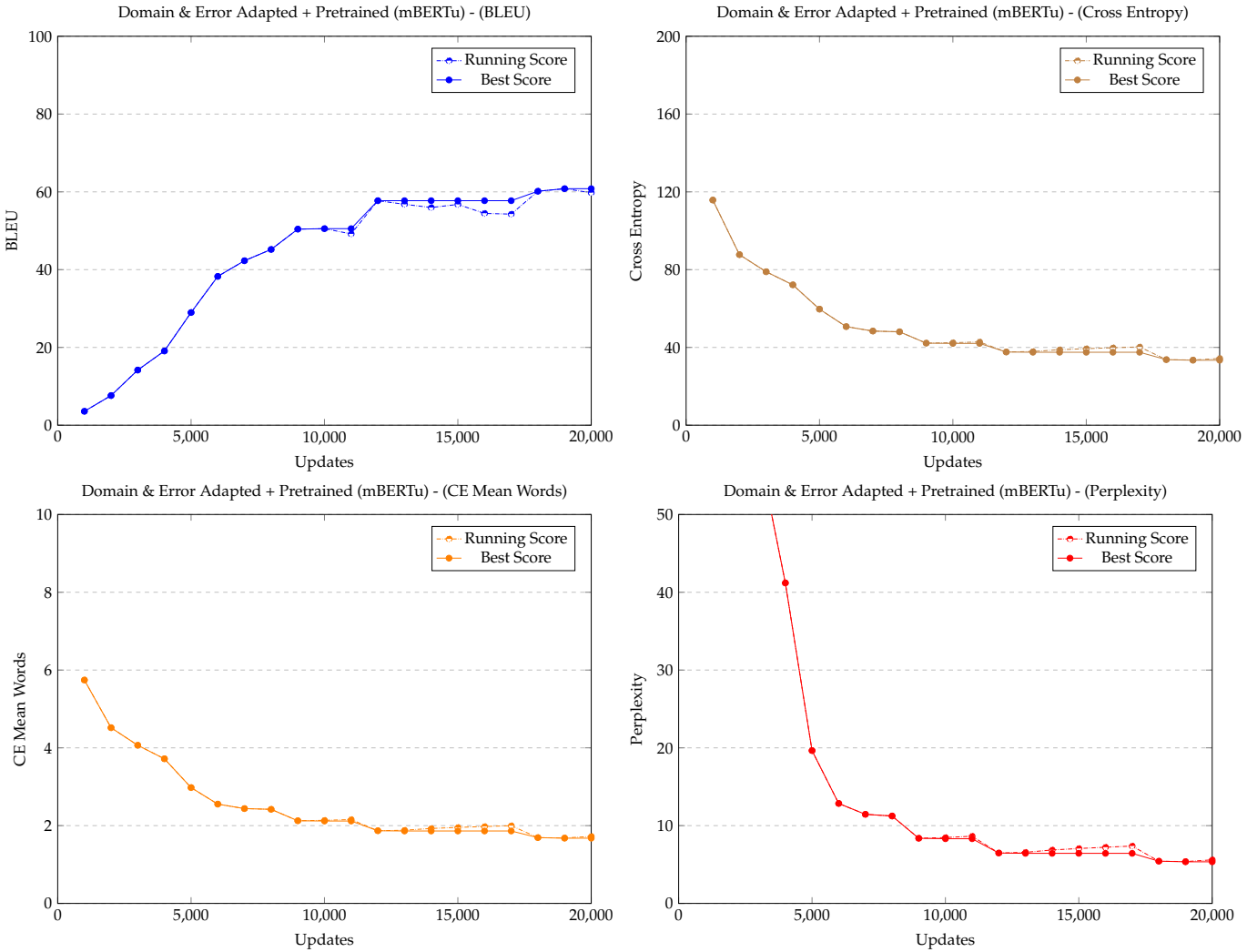

Figure 12: Domain & Error Adapted + Pretrained (mBERTu)

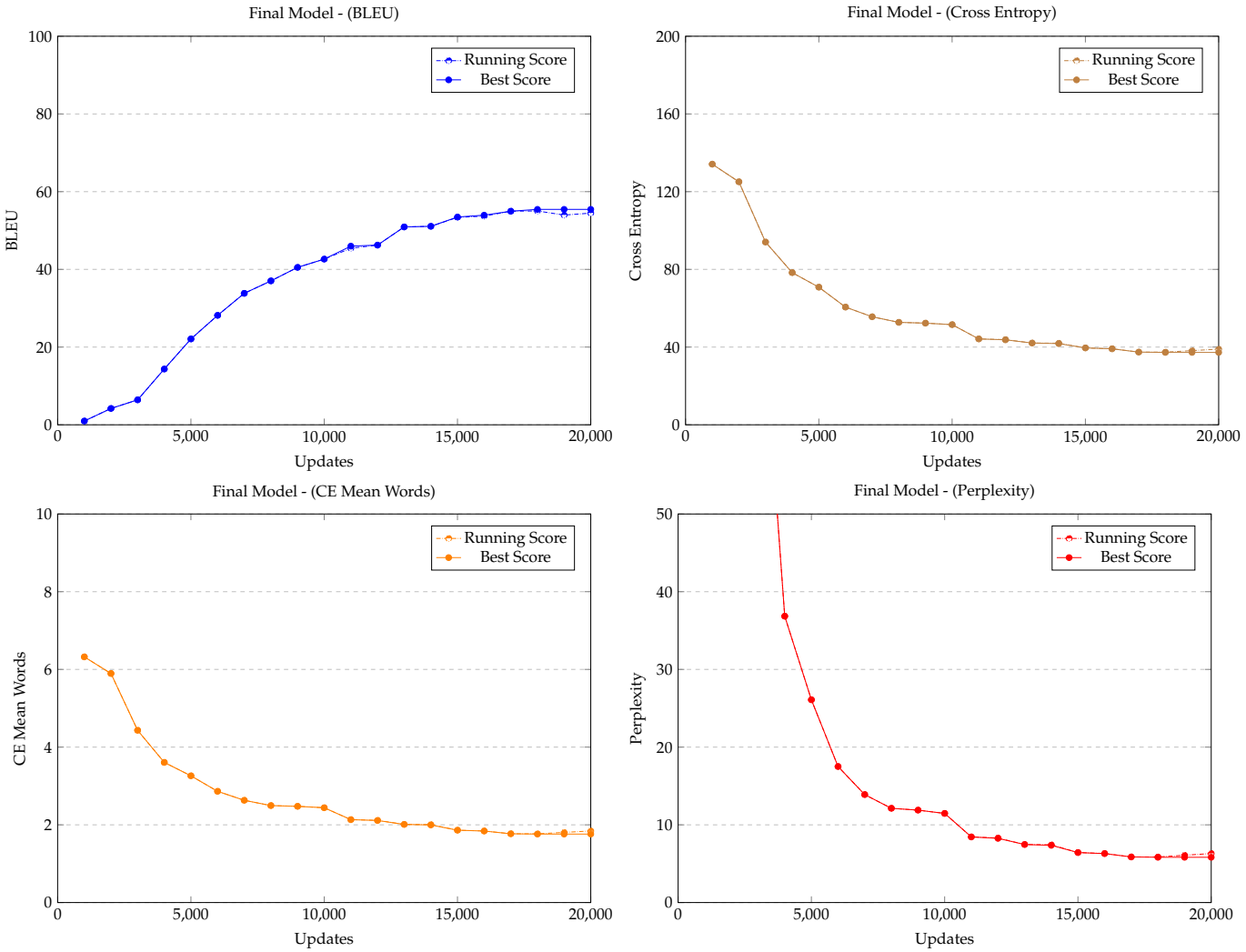

Figure 13: Final Model

User Manual

We encourage the reader to download the code and interface directly with the system described in this document. One can interact with the solution using the Google Colab Notebook titled *MGEC.ipynb*. Either download this document directly from the repository[1] or clone the repository. The repository contains the Notebook along with the source code and datasets.

If the Notebook file is downloaded directly, the first executable code cells will still clone the same repository because the aforementioned datasets and source code are prerequisites for the implementation. Any required third-party libraries are taken care of by the Notebook itself.

```
sudo apt install git-all
git init
apt-get install git-lfs
git lfs install
git clone https://github.com/AaronDebattista09/ICS5200.git
```

Listing 1: Bash Script for Cloning the Repository

[1] https://github.com/AaronDebattista09/ICS5200