Transcribing Natural Languages for the Deaf via Neural Editing Programs

Dongxu Li[1,2,†], **Chenchen Xu**[1,2], **Liu Liu**[3], **Yiran Zhong**[4,†],
Rong Wang[1], **Lars Petersson**[1,2], **Hongdong Li** [1]

[1]The Australian National University,
[2]DATA61-CSIRO,
[3]Huawei Cyberverse Lab,
[4]SenseTime
{dongxu.li@anu.edu.au, zhongyiran@sensetime.com}

Abstract

This work studies the task of *glossification*, of which the aim is to *transcribe* natural spoken language sentences for the Deaf (hard-of-hearing) community to ordered sign language glosses. Previous sequence-to-sequence language models trained with paired sentence-gloss data often fail to capture the rich connections between the two distinct languages, leading to unsatisfactory transcriptions. We observe that despite different grammars, glosses effectively simplify sentences for the ease of deaf communication, while sharing a large portion of vocabulary with sentences. This has motivated us to implement glossification by executing a collection of editing actions, *e.g.* word addition, deletion and copying, called *editing programs*, on their natural spoken language counterparts. Specifically, we design a new neural agent that learns to synthesize and execute editing programs, conditioned on sentence contexts and partial editing results. The agent is trained to imitate minimal editing programs, while exploring more widely the program space via policy gradients to optimize sequence-wise transcription quality. Results show that our approach outperforms previous glossification models by a large margin, improving the BLEU-4 score from 16.45 to 18.89 on RWTH-PHOENIX-WEATHER-2014T and from 18.38 to 21.30 on CSL-Daily.

Introduction

Glossification is the task of transcribing natural language sentences into glosses, the written form of sign languages (Johnston and Schembri 2007). Each sign gloss is usually a word that relates to a sign gesture. Glossification has important applications in automating deaf-hearing communication. Such a transcription step is considered as a necessary precursor to translating natural languages into videos of sign gestures (Stoll et al. 2020; Korte et al. 2020), thus alleviating the communication obstacles that the deaf and hard-of-hearing community members face and maximizing their performance in careers and other social engagement (Yin et al. 2021).

Gloss sequences follow their own ordering rules, and usually consist of fewer tokens than their natural language counterparts. For instance, the English sentence *"Do you like to watch baseball games?"* transcribes to American Sign Language (ASL) glosses *"baseball watch you like?"*. Such discrepancy in grammar requires glossification models to jointly represent both language sources in the embedding space and properly build mappings in-between, yielding a challenging sequence learning problem.

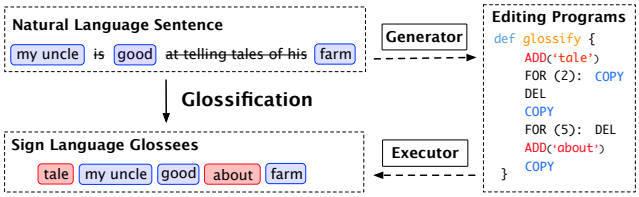

Figure 1: We study the problem of *glossification*, which aims to transcribe natural language sentences into sign language glosses. In particular, we design a neural agent that generates and executes *editing programs* on the natural language sentences to obtain glosses.

Previous glossification approaches (Stoll et al. 2020; Zhang and Duh 2021) take as input the natural language sentences and directly predict gloss sequences as output. However, gloss annotations are expensive to obtain due to the required expertise in sign language. As a result, the performances of models suffer from the limited amount of available parallel samples. Considering this data-scarce situation, we aim to improve glossification quality by effectively exploiting the syntactic connections between sentences and glosses.

Specifically, we notice that despite grammar gaps, glosses largely share a common vocabulary with sentences. In addition, glosses usually simplify sentences by keeping key content words and discarding those otherwise, for the ease of signing. These two observations lead us to the following perspective: instead of predicting glosses directly, our model derive transcriptions by making changes to the original sentence. In particular, we propose a new neural agent that learns to explicitly synthesize an ordered collection of editing actions for each input sentence, called an *editing program*. An editing program is composed of editing actions that remove, keep and add words based on the input sentence. By sequentially executing editing actions on the input sentence, the agent obtains glosses as output.

Our proposed agent consists mainly of two components that collaborate with each other: a *program generator* that at each time step predicts an editing action, and a *program executor* that applies actions on the input sentence to obtain

Corresponding authors: Dongxu Li and Yiran Zhong.

editing results, namely glosses. To facilitate the communication between the two modules, we further introduce a new attention mechanism, called *editing causal attention*, allowing the generator to attend to history partial glosses while preserving the auto-regressive property of the model.

Concretely, to learn the mapping from sentences to editing actions, the generator follows a typical transformer encoder-decoder structure, taking sentences as input. Different from previous approaches, instead of producing glosses directly, the generator synthesizes *partial editing programs* at each step as output. However, since the generator module receives editing labels as supervision, it learns mainly action labels yet not effectively leveraging the execution results, *i.e.* glosses, when making editing decisions. To address this issue, in our design we enable the executor module to communicate the glosses output as feedback to the generator to guide further action predictions. Particularly, at each step the executor first applies the predicted actions on the sentence to obtain partial gloss output, it then summarizes the glosses using an extra encoder. Finally, before the generator predicts an action, it communicates with the executor regarding the current editing results via the proposed editing causal attention. The editing causal attention is a variant of vanilla decoder attention, with the critical difference in that *the number of masked tokens is determined dynamically by the editing history*. With editing causal attention, we ensure the generator effectively attends to known partial editing results, thus better utilizing the program semantics for predicting further editing actions.

The agent is first trained to imitate minimal editing programs, obtained using a dynamic programming procedure similar to the Levenshtein distance algorithm (Schütze, Manning, and Raghavan 2008). However, we notice our agent gets overly-penalized due to the issue of program aliasing (Bunel et al. 2018): while multiple editing programs result in equivalent glosses yet all except the one provided as target are considered incorrect. To alleviate this issue, we adopt a policy-gradient method (Luo 2020) to reward the agent with semantically correct transcriptions, which we refer as *peer-critic*. The peer-critic method takes sequence-metrics as rewards, such as BLEU, and uses average rewards of peer samples as a baseline to reduce variance. Combining the imitation and reinforcement learning strategies, our agent achieves significantly better glossification results than existing methods.

Contributions. Our main contribution are as follows. (i) We introduce a novel glossification glossification method via sequential executions of editing actions. Such a formulation effectively enables to exploit syntactic connections between sentences and glosses, making our method stand out from peers that rely on conventional machine translation pipeline; (ii) We design a causal editing attention module, a variant of typical transformer decoder attention where the number of masked tokens are determined dynamically based on the produced partial glosses. In this way, we inform the generator of history execution results before making future decisions. (iii) We optimize our agent by imitating minimal ground-truth editing programs, while also encouraging it to explore wider program space to counteract the effect of program aliasing. (iv) Experiments on public datasets using sign languages from different regions show clearly preferable transcription quality from our system, both quantitatively and qualitatively through human evaluations with deaf involvement.

Related Work

Sign Language Recognition, Translation and Production. Most research works in sign language interpretation (Yin and Read 2020) aim at recognizing (Li et al. 2020a,c; Albanie et al. 2020; Min et al. 2021) and translating (Cihan Camgoz et al. 2018; Li et al. 2020b; Zhou et al. 2021a) visual sign gestures into sentences. However, to facilitate two-way deaf-hearing communication, it is also necessary to translate spoken sentences to sign gestures in videos or animations for the deaf (Korte et al. 2020). In this regard, the recent work (Stoll et al. 2020; Saunders, Camgoz, and Bowden 2020) first translates sentences to glosses, which are later used to produce continuous sign language videos (Saunders, Camgoz, and Bowden 2021). Our work follows this paradigm while for the first time, formalizing glossification as a standalone sign language interpretation task. Different from (Stoll et al. 2020) that adapts a neural machine translation approach, we propose to use editing labels to bridge the gap between the two linguistic sources, achieving superior glossification results.

Neural Program Synthesis. Program synthesis techniques aim to generate programs that satisfy given specifications (Gupta et al. 2020; Pu et al. 2020), either in natural languages or as a set of example inputs and desired outputs. The advantage of neural programs is their flexibility in modeling compositional structures in textual and visual data, thus are widely applied in various domains, including string manipulation (Reed and de Freitas 2016), sentence simplification (Dong et al. 2019), semantic parsing (Shin et al. 2019) and shape generation (Ellis et al. 2018; Tian et al. 2019). Inspired by these works, we use editing programs to exploit syntactic connections between sentences and glosses. Different from previous models that directly predict glosses (Stoll et al. 2020), we instead obtain glosses as the result of executing editing programs on their sentence counterparts. In this way, our model better utilizes relations between sentences and glosses by explicitly copying or removing words. In addition, editing programs define each glossification step as an action, thereby, they are easier to interpret than results from black-box sequence-to-sequence machine translation models.

Methodology

In this section, we present the main technical contributions of our proposed approach. First, we describe the ***definition of editing programs*** and also the way we construct ground-truth editing programs for training. Then, we detail the proposed ***architecture of the generator and executor*** modules. We also explain how these two modules communicate with each other, via a novel ***editing causal attention*** mechanism. Finally, we introduce the ***imitation and reinforcement learning strategies*** we adopt to train the glossification agent and alleviate the issue of program aliasing.

Editing Programs for Glossification

Problem definition. Given $\boldsymbol{x} = [x_1, ..., x_m] \in \mathcal{X}$ a natural language sentence with m words from a vocabulary V, and

Program	→	Statement; Program | $\varnothing$
AtomicStatement	→	ADD(Token) | DEL(PositionPointer) | COPY(PositionPointer) | SKIP
Statement	→	For(RepeatParam); AtomicStatement; EndFor | AtomicStatement
Token	→	t, $t \in V$, where V is the shared vocabulary of glosses and sentences.
PositionPointer	→	i, $i \in \mathbb{N}_{\geq 0}$
RepeatParam	→	r, $r \in \mathbb{N}^{+}$

Table 1: The syntax of domain specific language (DSL) used by editing programs in EBNF notation (Visser et al. 1997). We represent non-terminal symbols on the left and production rules on the right.

$\boldsymbol{y} = [y_1, ..., y_n] \in \mathcal{Y}$ the transcription with n glosses from the same shared vocabulary V, an *editing program synthesis* approach aims to compute an valid editing program $\boldsymbol{z} \in \mathcal{Z} : \mathcal{X} \rightarrow \mathcal{Y}$ which transforms $\boldsymbol{x}$ to $\boldsymbol{y}$, *i.e.*, $\boldsymbol{z}(\boldsymbol{x}) = \boldsymbol{y}$. Note that $\boldsymbol{z}$ is not unique and there may exist multiple programs satisfying the input-output specification.

Definition of editing programs. The syntax of the domain specific language (DSL) for editing programs is given in Table 1. Specifically, each program $\boldsymbol{z}$ contains a variable number of program statements, including four atomic statements (or *editing actions*) and a looping construct.

In terms of the atomic statements, we define (i) ADD(w), which selects a token w from the vocabulary and appends w to the gloss sequence $\boldsymbol{y}$. The sentence $\boldsymbol{x}$ remains intact when an ADD action is applied; (ii) DEL(k), which removes the word x_k from the sentence; (iii) COPY(k), which keeps the word x_k from the sentence and appends it to $\boldsymbol{y}$, for example; (iv) SKIP, which discards remaining sentence tokens and completes the glossification procedure.

We also introduce a looping construct, For(r), that applies an atomic statement for r repetitions. Benefits for including the For statement in the editing program are threefold. First, it captures the regularity when several consecutive words in the sentence are handled by the same atomic statement. Second, it reduces the length of programs and eases the difficulty during long-range inference. Third, since gloss sequences are usually shorter than their sentence counterparts, the number of DEL actions to apply is larger than other actions. In this regard, the For statement alleviates the challenge of synthesizing programs with imbalanced action classes.

Minimal editing program. As aforementioned, our agent learns to predict editing actions to derive glosses. To achieve this, we provide expert editing programs to demonstrate program syntax and semantics to the agent. In this regard, we first design a dynamic programming algorithm to compute *minimal editing programs* for each sentence-gloss pair.

Given a sentence-gloss pair, a minimal editing program is the one that consists of the least number of ADD and DEL actions to transform a sentence to its gloss transcription. Specifically, we adapt the procedure to compute Levenshtein distances (Schütze, Manning, and Raghavan 2008) while discarding the substitution actions, thereby avoiding the quadratic growth of the number of editing actions with the vocabulary size. We first compute the minimal editing distance (Schütze, Manning, and Raghavan 2008) between the sentence $\boldsymbol{x}$ and the glosses $\boldsymbol{y}$, and then extract actions from the trajectory with the minimal editing distance. When there existing multiple trajectories of the same number of editing actions, we priortize ADD over DEL to ensure the uniqueness of the minimal editing program. Finally, we compress the identical consecutive actions by the For statement.

Neural Editing Program Synthesis and Execution

An overview of our glossification model is shown in Fig. 2. Given a sentence in natural language $\boldsymbol{x} = [x_1, ..., x_m]$, our model predicts an editing program $\boldsymbol{z}$ to glossify $\boldsymbol{x}$ to $\boldsymbol{y} = [y_1, ..., y_n]$ by modeling the conditional distribution $P(\boldsymbol{z}|\boldsymbol{x})$,

$$P(\boldsymbol{z}|\boldsymbol{x}) = \prod_{t=1}^{|\boldsymbol{z}|} P(z_t|\, \boldsymbol{x},\, y_{1:j_{t-1}},\, z_{1:t-1}). \tag{1}$$

Particularly, at each time step t, we use a *generator* to predict the next statement z_t considering (i) the natural language sentence $\boldsymbol{x}$; (ii) history gloss outputs $y_{1:j_{t-1}}$, where j_{t-1} denotes the length of output glosses until time $t-1$; and (iii) history editing statements $z_{1:t-1}$ until time $t-1$. To effectively utilize history editing results, we also design an *executor* taking $\boldsymbol{x}$ and partial statements $z_{1:t-1}$ as input, then derives and summarizes history glosses $y_{1:j_{t-1}}$.

Since our editing program generation procedure is partially conditioned on the history execution results, we first introduce the program executor followed by the generator and their communication mechanism in-between.

Program Executor. Given a natural language sentence $\boldsymbol{x}$ and a synthesized (partial) program $z_{1:t-1}$, our program executor first applies the editing statements on the sentence $\boldsymbol{x}$ to obtain the gloss output $y_{1:j_{t-1}}$. To achieve this, we maintain an *executor pointer* k that holds the index of the current word to edit in $\boldsymbol{x}$. Before the agent applies a COPY or DEL action, it determines the word to edit based on the pointer value. Rules to update k are as follows. The pointer k starts from x_1, the first element of $\boldsymbol{x}$. Each time a DEL or COPY action is executed, k moves to the next position of $\boldsymbol{x}$ and points to x_{k+1}, indicating x_k is either kept in the glosses $\boldsymbol{y}$ or discarded during the execution. When an ADD action is applied, k remains unchanged since no editing happens in the sentence $\boldsymbol{x}$. On encountering a For statement, k moves forward by r positions and points to x_{k+r}. Glosses $y_{1:j_{t-1}}$ are then obtained by executing the partial program $z_{1:t-1}$ sequentially on the sentence x.

After obtaining the partial glosses, the executor summarizes the output $y_{1:j_{t-1}}$ and prepares for communicating this execution result with the generator for future predictions. To achieve this, the executor represents the glosses in the embedding space. In particular, we feed glosses $y_{1:j_{t-1}}$ to a number

In the rest of the manuscript, we omit the action parameters when it is unambiguous from the context.

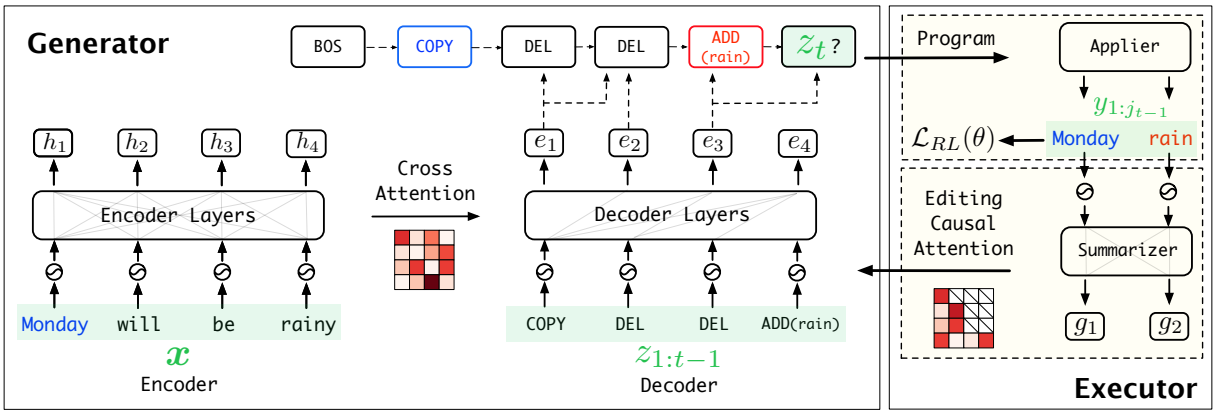

Figure 2: Our glossification model consists of two main modules: a *generator* and an *executor*. The generator predicts editing actions from the input sentence. The executor applies the program to derive glosses and provides execution feedback, which is then communicated with the generator via two channels: the editing causal attention mechanism and the peer-critic objective.

of Transformer encoder layers (Vaswani et al. 2017) to obtain their hidden embeddings $g_{1:j_{t-1}}$. For the l-th encoder layer:

$$g_1^{(l+1)}, ..., g_{j_{t-1}}^{(l+1)} = \begin{cases} E_{y_1} + P_1, ..., E_{y_{j_{t-1}}} + P_{j_{t-1}}, & l = 1, \\ \text{EncoderLayer}_l(g_1^{(l)}, ..., g_{j_{t-1}}^{(l)}), & l > 1; \end{cases} \quad (2)$$

where $E \in \mathbb{R}^{|V| \times d_{\text{model}}}$ and $P \in \mathbb{R}^{L_{\max} \times d_{\text{model}}}$ are look-up tables that map the i-th gloss y_i to its token embedding and sinusoidal positional encoding (Vaswani et al. 2017), respectively, with $L_{\max}$ the maximal input lengths and d_{model} the hidden dimension. The EncoderLayer$(\cdot)$ is composed of self-attention layers and position-wise feed-forward networks to capture pairwise dependencies among feature embeddings.

Program Generator. We formulate the program generation procedure as a sequential prediction problem and employ an encoder-decoder model for generating programs. In particular, the encoder of the generator takes as input the natural language sentence $\boldsymbol{x}$, and represents each word x_i in the embedding space as $h_i \in \mathbb{R}^{d_{\text{model}}}$, similar to the summarization procedure in Eq. (2). The decoder models the conditional probability $P(\boldsymbol{z}|\boldsymbol{x})$ as in Eq. (1), while at the same time communicating with the executor regarding the history gloss output using an editing causal attention mechanism.

Specifically, at time step t, given the partial program $z_{1:t-1}$ and the hidden sentence representations $\boldsymbol{h} = [h_1, h_2, ..., h_m]$ with m the input length, the decoder first computes the representation of each statement in the editing history $e_i \in \mathbb{R}^{d_{\text{model}}}$ using Transformer decoder layers (Vaswani et al. 2017).

$$e_1^{(l'+1)}, ..., e_{t-1}^{(l'+1)} = \begin{cases} E_{z_1} + P_1, ..., E_{z_{t-1}} + P_{t-1}, & l' = 1, \\ \text{DecoderLayer}_{l'}(e_1^{(l')}, ..., e_{t-1}^{(l')}, & \\ \qquad h_1, ..., h_m), & l' > 1; \end{cases} \quad (3)$$

where l' is the index of decoder layers. The token embedding and positional encoding are similar to those in the encoder except that they apply to the editing history. In addition to the two sub-layers as in the encoder, the decoder consists of an extra sub-layer, which performs attention over the sentence representation $\boldsymbol{h}$ from the encoder. In this way, the decoder considers the full context of the sentence input as well as the past editing statements when making further predictions.

Editing causal attention for execution-guided generation. As described above, the program generator models the relation between the program $\boldsymbol{z}$ and the sentence $\boldsymbol{x}$. However, there exists a clear gap in such a design that the partial gloss output $y_{1:t-1}$ is not fully utilized. In other words, the generator largely ignores the semantics of the editing programs. This is less desirable as the partial execution states provide useful guidance for predicting future program statements (Chen, Liu, and Song 2019). In particular for the glossification task, partial glosses provide helpful contexts for predicting the next editing action to take. In light of this observation, we develop an *editing causal attention*, a mechanism that effectively allows the generator to take into account history glosses in the followup program generation process.

The editing causal attention is a variant of masked multi-head attention (Vaswani et al. 2017). Yet differently, since the model produces one sign gloss only when an `ADD` or `COPY` operation is applied, while the length the gloss output remains unchanged for `DEL` actions, the number of masks is dynamically determined based on the predicted editing history $y_{1:j_{t-1}}$. In this way, we effectively prevent editing actions from peeking future glosses and preserve the auto-regressive property of the generator, thereby, preserving the auto-regressive property of the generator model.

Specifically, we maintain a *generator pointer*, which points at the gloss sequence $\boldsymbol{y}$ and records j_{t-1}, the length of the current gloss sequence. The pointer is initialized as zero, indicating that the history gloss is empty. When the executor applies an `ADD` or `COPY` action, the generator pointer moves forward by one position, suggesting that one more gloss is produced. On encountering a `For` statement, the pointer moves forward by the number of repetitions. When a `DEL` action is applied, the pointer remains unchanged since no new gloss is produced. During the teacher-forcing training, when predicting the editing action z_t at time step t, gloss positions larger than j_{t-1} are masked.

We add an editing causal attention layer on top of the

last decoder layer. Specifically, the communication between the generator and the executor is achieved via a masked scaled dot-product attention $\mathcal{G}_{\text{attn}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{Q}\mathbf{K}^T/\sqrt{d})\mathbf{V}$, where $\mathbf{Q}$ represents transformed features of $e_{1:t-1}$ via a feed-forward layer and $\mathbf{K}$, $\mathbf{V}$ are those of $g_{1:j_{t-1}}$, d is the feature dimension of vectors in $\mathbf{K}$. The attention matrix from the $\text{softmax}(\cdot)$ function is masked such that illegal positions of future glosses are filled with $-\infty$, with the number of masked tokens determined as explained beforehand. Finally, we add a single-layer feed-forward network as the classifier to predict the program statement z_t. A visual example of learned editing causal attention map is shown in Fig. 4.

Handling looping statements. Generally, `For` statements in programming languages allow nested loops. However, this creates additional complications in constructing deterministic minimal editing programs as supervision as well as during program generation. To ease this complexity, we guide the generator with a restrictive usage of `For` constructs, allowing only atomic statements to repeat. During the program generation, this adaption effectively only requires the model to predict an atomic statement to apply, along with an integer denoting the number of repetitions.

Learning to Imitate and Explore

We provide the agent with initial task knowledge in an *imitation learning* strategy, where the agent takes the teacher action z_t^* at each time step to efficiently learn to imitate the minimal editing programs. However, in practice, we notice the agent is overly-penalized due to the issue of program aliasing (Bunel et al. 2018). Namely, multiple editing programs result in equivalent glosses yet all except the one provided as target are considered incorrect. An obvious example as such in our case is that to `COPY` a token x_k from the sentence $\boldsymbol{x}$ is equivalent to predict an `ADD` action with the same token x_k.

Inspired by recent works in visual captioning (Rennie et al. 2017; Luo 2020), we propose to combine the imitation learning (IL) objective with a policy gradient method (RL) that rewards the agent for correct gloss outputs. In this way, we encourage the agent to not only exploit the expert knowledge in minimal editing programs, but also to explore more widely the program space for semantically-equivalent programs. Specifically, we compute the reward by evaluating the BLEU-4 score of the generated glosses with respect to the corresponding ground-truth glosses. The goal of RL objective is to minimize the negative expected reward. The overall optimization objective combines IL and RL objectives as follows:

$$\mathcal{L}(\theta) = \underbrace{\lambda \sum_{t=1}^{T} -z_t^* \log(q_t)}_{\mathcal{L}_{IL}(\theta)} \underbrace{-\mathbb{E}_{\tilde{\boldsymbol{z}} \sim P_\theta(\boldsymbol{z}|\boldsymbol{x})}[r(\tilde{\boldsymbol{z}})]}_{\mathcal{L}_{RL}(\theta)}, \quad (4)$$

with q_t the output probability of ground-truth editing label z_t^*, $\tilde{\boldsymbol{z}} = [\tilde{z}_1, ..., \tilde{z}_t]$ the editing statement sampled from the model at time step t, $P_\theta(\boldsymbol{z}|\boldsymbol{x})$ the glossification model (generator and executor) parameterized by θ and $r(\cdot)$ the reward function, respectively. The hyperparameter λ balances between the imitation learning and reinforcement learning objectives. Since the $\mathcal{L}_{RL}(\theta)$ term in Eq. (4) is non-differentiable *w.r.t.* θ, we then use the REINFORCE algorithm (Williams 1992) to compute the gradient with Monte-Carlo sampling,

$$\nabla_\theta \mathcal{L}_{RL}(\theta) = -r(\tilde{\boldsymbol{z}}) \nabla_\theta \log P_\theta(\tilde{\boldsymbol{z}}|\boldsymbol{x}). \quad (5)$$

In practice, we approximate the expected rewards with the average of $K = 5$ samples from $P_\theta(\boldsymbol{z}|\boldsymbol{x})$. To reduce the variance of the estimation, we follow the method as suggested in (Luo 2020), and further compute the reward function relative to a baseline b, resulting the reward as an advantage function. For each sample, its baseline is the average reward of the remaining $K - 1$ peer samples. In the sequel, we refer this RL objective as the *peer-critic* objective for simplicity. We refer interested readers to (Luo 2020) for more details.

Experiments

Implementation Details and Experiment Setup

Implementation. We implement our model with the framework FAIRSEQ (Ott et al. 2019) in PYTORCH (Paszke et al. 2019). To represent texts in the feature space, we use pretrained German and Chinese embeddings (Joulin et al. 2016). The generator consists of three encoder layers and one decoder layer; the executor consists of a single encoder layer. We adopt ten parallel heads in all the multi-head attention modules to learn diverse patterns. During the training, we warm up the agent with the imitation learning objective for 25 epochs so that it learns the syntax and semantic rules of DSL efficiently. Then we add the peer-critic objective to encourage the agent to explore more widely the program space with semantically correct statements. We optimize our model using the Adam optimizer (Kingma and Ba 2014), with an initial learning rate of 10^{-4} and a weight decay of 10^{-4}. All the hyperparameters are selected using the validation partition. We train our networks for 150 epochs, which is sufficient for all the models to converge, each taking around 30 hours on a single NVIDIA P100 GPU. Code will be made public.

Datasets. We evaluate our glossification approach on two widely-used public datasets, including RWTH-PHOENIX-Weather 2014T (RPWT) dataset (Cihan Camgoz et al. 2018) and CSL-Daily (Zhou et al. 2021b), the only two existing datasets that provide parallel sentence-gloss annotations for large-scale training and inference.

Specifically, **RPWT** has glosses in German Sign Language (GSL/DGS) and sentences in German, while **CSL-Daily** contains Chinese Sign Language (CSL) glosses paired with sentences in Chinese. On both datasets, we follow the public data partition protocol, with $7,096$, 519, 642 sentence-gloss pairs for training, validation and testing on RPWT and $18,401$, $1,077$ and $1,176$ pairs, respectively for CSL-Daily. Using these two datasets, we validate and demonstrate the potentials of our approach to generalize to sign languages from different geographic regions.

Metrics. Our method achieves glossification by synthesizing editing programs. To validate the proposed method, we evaluate two aspects of the synthesized editing programs.

- **Program alignment.** The predicted program is a *perfect alignment* if it is identical to the minimal editing program.

Methods	PER ↓	R-L ↑	B-3 ↑	B-4 ↑
Previous Glossification Models				
Text2Sign	-	48.10	21.54	15.26
Zhang et al	-	49.19	23.03	16.45
Related Sequence Learning Models				
EditNTS	-	46.62	20.23	14.75
CopyNet†	-	48.41	21.74	15.86
Baseline	56.51	47.07	22.44	16.01
Edit-Att	**55.24**	49.66	24.93	18.07
Edit-Att + $\mathcal{L}_{RL}$	55.56	**49.91**	**25.51**	**18.89**

(a) RWTH-PHOENIX-WEATHER-2014T (RPWT)

Methods	PER ↓	R-L ↑	B-3 ↑	B-4 ↑
Previous Glossification Models				
Text2Sign†	-	49.19	25.46	18.80
Zhang et al†	-	46.61	26.09	18.38
Related Sequence Learning Models				
EditNTS	-	51.89	28.21	19.88
CopyNet†	-	**52.84**	28.77	20.24
Baseline	48.29	50.31	25.91	17.97
Edit-Att	**47.16**	52.31	28.82	20.56
Edit-Att + $\mathcal{L}_{RL}$	48.30	52.78	**29.70**	**21.30**

(b) CSL-Daily

Table 2: Results of quantitative comparisons. We show metrics ROUGE-L (R-L), BLEU-3 (B-3) and BLEU-4 (B-4). We report official results or results from officially released models when possible, and use (†) to denote our reproduced results otherwise.

When the alignment is not perfect, we compute the *program error rate* (PER) to measure the alignment between the synthesized program and the minimal editing program, which adapts word error rate (WER) (Hinton et al. 2012) used in speech recognition research and computes alignment between editing statements.

- **Generalization.** The predicted program is a *generalization* if it satisfies the input-output specification (Chen, Liu, and Song 2019). For glossification, this requires comparing the predicted glosses with the ground-truth ones. In this regard, we use the BLEU (Papineni et al. 2002) and ROUGE-L (Lin and Och 2004) scores, two commonly adopted measurements for sequences. BLEU-n measures the precision of the sequence up to n-gram. ROUGE-L measures the F1 score based on the longest common subsequences between predictions and ground-truth glosses.

Method Evaluation

The aim of the evaluation is three-fold. First, we demonstrate the advantage of the proposed glossification approach via synthesizing editing programs. This is achieved by comparing our method (with ablations) with previous glossification and related sequence learning methods quantitatively. Second, we take RPWT as an example and analyze effects of important components and design choices. Third, we validate that our approach improves the transcription quality for the deaf community. We achieve this by human evaluations on the CSL-Daily dataset with deaf involvement.

Competing Methods. We compare our approaches with two groups of competing methods. (i) Previous glossification approaches, including Text2Sign (Stoll et al. 2020) and Zhang et al (Zhang and Duh 2021). Both works adopt a conventional encoder-decoder architecture, taking sentences as input and glosses as output. In particular, Text2Sign uses GRU (Chung et al. 2014) as the underlying model and Zhang et al. use Transformers (Vaswani et al. 2017). (ii) Related methods from other sequence learning tasks. Considering the overlap between sentences and glosses, we compare with CopyNet (Gu et al. 2016), a general sequence learning method that selectively replicates segments from the input to the output. Such copying mechanism proves desirable for the glossification task. Observing that glosses are usually shorter than their sentence counterpart, we also compare with a text simplification model EditNTS (Dong et al. 2019) that simplifies complex sentences by explicit editing operations.

Quantitative comparison. Results are shown in Table 2. The row of *baseline* stands for the model without either the editing causal attention or the peer-critic objective. As indicated in the table, our approach consistently outperforms previous glossification approach by a large margin. Compared with approaches that directly predict glosses (Text2Sign, Zhang et al and CopyNet), our approach exploits the linguistic relation between sentences and glosses. In this way, our agent effectively reuses content words from the sentence, thus obtaining superior transcription results. Compared with EditNTS, our model follows a Transformer architecture and is equipped with the novel editing causal attention module. The editing attention allows the generator to take into account of partial glosses as the feedback of program executions more flexibly than the hard attention in EditNTS. A visual example of the editing causal attention is shown in Fig. 4. In addition, we observe that the peer-critic objective adversely impacts PER yet results in better sequence-level metrics. This indicates that beyond following the minimal editing programs, our agent also searches for semantically correct editing programs. These validate our motivations for the peer-critic objective.

Qualitative results. Figure 3 shows an example transcription produced by our model on RPWT. Our model succeeds in providing high-quality gloss sequences that match the ground-truth. Note that the example well demonstrates the issue of program aliasing: the minimal editing program consists of a `COPY` operation to obtain the word *wechselhaft*, while our model decides to apply an `ADD` action alternatively. To obtain

λ	PER	R-L	B-4
0.1	54.84	49.63	18.33
0.5	**55.56**	**49.91**	**18.89**
1.0	55.68	50.46	18.60

Reward	PER	R-L	B-4
R-L	55.48	**50.17**	18.21
B-4	55.56	49.91	**18.89**
R+B	**55.18**	49.77	18.17

Table 3: Results with different λ values (left) and with different metrics as rewards (right) on the RPWT dataset.

Sentence	montag ~~und~~ dienstag ~~wechselhaft hier und da zeigt~~ ~~sich aber~~ auch die sonne . (monday and tuesday changeable here and there but the sun also shows up .)
Prediction	COPY DEL COPY ADD(wechselhaft) ADD(mal) DEL5 DEL2 COPY COPY COPY SKIP montag dienstag wechselhaft mal auch die sonne
Reference	COPY DEL COPY COPY ADD(mal) DEL5 DEL COPY DEL COPY SKIP montag dienstag wechselhaft mal auch sonne

Figure 3: Example outputs of our glossification approach on the testing set of RPWT dataset. The *prediction* rows show the synthesized programs and execution results. The *reference* rows show minimal editing programs and ground-truth glosses. We highlight tokens to COPY in blue, to ADD in red and tokens to delete as to ~~strikeout~~. We show correctly glossified 1-grams in green. We add an integer n after an action as an abbreviation for For statements with n repetitions.

	Correctness ↓			**Adequacy** ↓			**B-4** ↑		
	Ours	CN	ZH	Ours	CN	ZH	Ours	CN	ZH
Short	**1.45**	1.95	2.60	**1.50**	1.80	2.70	**24.38**	22.97	21.27
Medium	**1.50**	2.15	2.35	**1.70**	1.90	2.40	**21.52**	20.38	18.12
Long	1.95	**1.90**	2.15	**1.85**	**1.85**	2.30	**19.77**	19.42	17.87

Table 4: Average ranking for Correctness, Adequacy by two deaf volunteers on the CSL-Daily dataset on our methods, CopyNet (CN) and Zhang et al (ZH). We select ten short (<8 characters), medium (8∼15 characters), long sentences each (>15 characters). BLEU-4 is also shown for the samples.

the correct glosses, our model then learns to apply one more deletion operation than the minimal editing program, in order to remove *wechselhaft* from the original sentence.

Model analysis and discussions. (i) Without introducing For statements, the best BLEU-4 score drops to 18.13. This is because the number of DEL actions in the minimal editing programs increases, worsening the action imbalance issue. (ii) We report the experiment results using different λ values between multi-tasking objectives in Table 3. It shows that introducing the peer-critic objective helps to improve the sequence metrics while does not guarantee fewer errors in modeling ground-truth programs. (iii) We also experiment with using ROUGE-L scores and a combination of ROUGE-L and BLEU-4 as the reward function for the peer-critic objective. The best model achieves 50.17 in ROUGE-L score and 18.21 in BLEU-4, validating the effect of peer-critic. We report results using the BLEU scores as the objective because it provides more visible improvement on different metrics.

Human evaluation with CSL users. With the help of two judges, we report human evaluation results in Table 4. Both judges are from the deaf community and are native CSL users. During the evaluation, we ask the judges to rank the models based on (i) correctness: whether glosses follow correct grammars? This is necessary as regional sign languages dialect may lead to correct yet different transcriptions from the ground-truth; (ii) adequacy: how much intent from the original sentences is preserved? The average rankings from the two judges show that our system is overall preferred. However, transcription becomes harder when sentence lengths grow. As a result, the difference between models become relatively less evident. It is our future work to improve glossification performance especially on long sentence inputs.

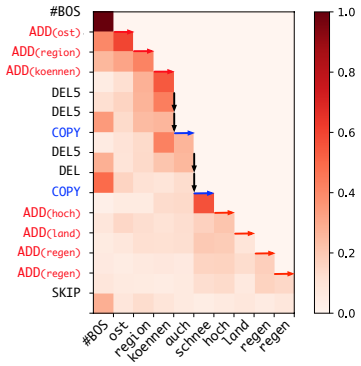

Figure 4: A visual example of the editing causal attention. To prevent the generator from peeking at future glosses during training, we reveal the next gloss only when either a COPY or ADD action is executed.

Conclusion

In this work, we have proposed a new approach that transcribes natural language sentences into sign language glosses. Instead of directly predicting glosses as output, our model learns to derive glosses as the result of editing the input natural language sentences. This is achieved by a generator module which synthesizes the editing programs and an executor module that performs the actions. These two modules communicate about the execution results via a new editing causal attention mechanism. To account for the program aliasing issue, our agent learns to imitate the ground-truth action sequences while at the same time exploring the wider program space via a policy-based objective. Our approach yields significantly better transcription quality on commonly-adopted public sign language datasets, verified quantitatively and qualitatively by human evaluation with deaf involvement. In addition, editing programs are more explainable than otherwise "black-box" type sequence-to-sequence models, offering a new perspective to the glossification task and visual sign language research in general.

Acknowledgements

This research is funded in part by ARC-Discovery grants (DP 190102261 and DP220100800) to HL. We thank reviewers and chairs for their constructive feedback and suggestion.

References

Albanie, S.; Varol, G.; Momeni, L.; Afouras, T.; Chung, J. S.; Fox, N.; and Zisserman, A. 2020. BSL-1K: Scaling up co-articulated sign language recognition using mouthing cues. In *European Conference on Computer Vision*, 35–53. Springer.

Bunel, R.; Hausknecht, M.; Devlin, J.; Singh, R.; and Kohli, P. 2018. Leveraging Grammar and Reinforcement Learning for Neural Program Synthesis. In *International Conference on Learning Representations*.

Chen, X.; Liu, C.; and Song, D. 2019. Execution-Guided Neural Program Synthesis. In *International Conference on Learning Representations*.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*.

Cihan Camgoz, N.; Hadfield, S.; Koller, O.; Ney, H.; and Bowden, R. 2018. Neural sign language translation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 7784–7793.

Dong, Y.; Li, Z.; Rezagholizadeh, M.; and Cheung, J. C. K. 2019. EditNTS: An Neural Programmer-Interpreter Model for Sentence Simplification through Explicit Editing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 3393–3402.

Ellis, K.; Ritchie, D.; Solar-Lezama, A.; and Tenenbaum, J. 2018. Learning to Infer Graphics Programs from Hand-Drawn Images. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Gu, J.; Lu, Z.; Li, H.; and Li, V. O. 2016. Incorporating Copying Mechanism in Sequence-to-Sequence Learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1631–1640.

Gupta, K.; Christensen, P. E.; Chen, X.; and Song, D. 2020. Synthesize, Execute and Debug: Learning to Repair for Neural Program Synthesis. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 17685–17695. Curran Associates, Inc.

Hinton, G.; Deng, L.; Yu, D.; Dahl, G. E.; Mohamed, A.-r.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T. N.; et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6): 82–97.

Johnston, T.; and Schembri, A. 2007. *Australian Sign Language (Auslan): An introduction to sign language linguistics*. Cambridge University Press.

Joulin, A.; Grave, E.; Bojanowski, P.; Douze, M.; Jégou, H.; and Mikolov, T. 2016. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Korte, J.; Bender, A.; Gallasch, G.; Wiles, J.; and Back, A. 2020. A plan for developing an Auslan communication technologies pipeline. In *European Conference on Computer Vision*, 264–277. Springer.

Li, D.; Rodriguez, C.; Yu, X.; and Li, H. 2020a. Word-level Deep Sign Language Recognition from Video: A New Large-scale Dataset and Methods Comparison. In *The IEEE Winter Conference on Applications of Computer Vision*, 1459–1469.

Li, D.; Xu, C.; Yu, X.; Zhang, K.; Swift, B.; Suominen, H.; and Li, H. 2020b. TSPNet: Hierarchical Feature Learning via Temporal Semantic Pyramid for Sign Language Translation. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 12034–12045. Curran Associates, Inc.

Li, D.; Yu, X.; Xu, C.; Petersson, L.; and Li, H. 2020c. Transferring Cross-domain Knowledge for Video Sign Language Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Lin, C.-Y.; and Och, F. J. 2004. Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, 605. Association for Computational Linguistics.

Luo, R. 2020. A Better Variant of Self-Critical Sequence Training. *arXiv preprint arXiv:2003.09971*.

Min, Y.; Hao, A.; Chai, X.; and Chen, X. 2021. Visual Alignment Constraint for Continuous Sign Language Recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 11542–11551.

Ott, M.; Edunov, S.; Baevski, A.; Fan, A.; Gross, S.; Ng, N.; Grangier, D.; and Auli, M. 2019. fairseq: A Fast, Extensible Toolkit for Sequence Modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

Papineni, K.; Roukos, S.; Ward, T.; and Zhu, W.-J. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, 311–318. Association for Computational Linguistics.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32*, 8024–8035. Curran Associates, Inc.

Pu, Y.; Ellis, K.; Kryven, M.; Tenenbaum, J.; and Solar-Lezama, A. 2020. Program Synthesis with Pragmatic Communication. *Advances in Neural Information Processing Systems*, 33.

Reed, S.; and de Freitas, N. 2016. Neural Programmer-Interpreters. In *International Conference on Learning Representations*.

Rennie, S. J.; Marcheret, E.; Mroueh, Y.; Ross, J.; and Goel, V. 2017. Self-critical sequence training for image captioning.

In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 7008–7024.

Saunders, B.; Camgoz, N. C.; and Bowden, R. 2020. Progressive transformers for end-to-end sign language production. In *European Conference on Computer Vision*, 687–705. Springer.

Saunders, B.; Camgoz, N. C.; and Bowden, R. 2021. Mixed SIGNals: Sign Language Production via a Mixture of Motion Primitives. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 1919–1929.

Schütze, H.; Manning, C. D.; and Raghavan, P. 2008. *Introduction to information retrieval*, volume 39. Cambridge University Press Cambridge.

Shin, E. C.; Allamanis, M.; Brockschmidt, M.; and Polozov, A. 2019. Program Synthesis and Semantic Parsing with Learned Code Idioms. *Advances in Neural Information Processing Systems*, 32: 10825–10835.

Stoll, S.; Camgoz, N. C.; Hadfield, S.; and Bowden, R. 2020. Text2Sign: towards sign language production using neural machine translation and generative adversarial networks. *International Journal of Computer Vision*, 128(4): 891–908.

Tian, Y.; Luo, A.; Sun, X.; Ellis, K.; Freeman, W. T.; Tenenbaum, J. B.; and Wu, J. 2019. Learning to Infer and Execute 3D Shape Programs. In *International Conference on Learning Representations*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, 5998–6008.

Visser, E.; et al. 1997. *Syntax definition for language prototyping*. Ponsen & Looijen.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4): 229–256.

Yin, K.; Moryossef, A.; Hochgesang, J.; Goldberg, Y.; and Alikhani, M. 2021. Including Signed Languages in Natural Language Processing. *arXiv preprint arXiv:2105.05222*.

Yin, K.; and Read, J. 2020. Better sign language translation with STMC-transformer. In *Proceedings of the 28th International Conference on Computational Linguistics*, 5975–5989.

Zhang, X.; and Duh, K. 2021. Approaching Sign Language Gloss Translation as a Low-Resource Machine Translation Task. In *Proceedings of the 18th Biennial Machine Translation Summit*.

Zhou, H.; Zhou, W.; Qi, W.; Pu, J.; and Li, H. 2021a. Improving Sign Language Translation with Monolingual Data by Sign Back-Translation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhou, H.; Zhou, W.; Qi, W.; Pu, J.; and Li, H. 2021b. Improving Sign Language Translation with Monolingual Data by Sign Back-Translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 1316–1325.