TAPIR: Learning Adaptive Revision for Incremental Natural Language Understanding with a Two-Pass Model

Patrick Kahardipraja[1] **Brielen Madureira**[1] **David Schlangen**[1,2]
[1]Computational Linguistics, Department of Linguistics
University of Potsdam, Germany
[2]German Research Center for Artificial Intelligence (DFKI), Berlin, Germany
{kahardipraja,madureiralasota,david.schlangen}@uni-potsdam.de

Abstract

Language is by its very nature incremental in how it is produced and processed. This property can be exploited by NLP systems to produce fast responses, which has been shown to be beneficial for real-time interactive applications. Recent neural network-based approaches for incremental processing mainly use RNNs or Transformers. RNNs are fast but monotonic (cannot correct earlier output, which can be necessary in incremental processing). Transformers, on the other hand, consume whole sequences, and hence are by nature non-incremental. A *restart-incremental* interface that repeatedly passes longer input prefixes can be used to obtain partial outputs, while providing the ability to revise. However, this method becomes costly as the sentence grows longer. In this work, we propose the Two-pass model for AdaPtIve Revision (TAPIR) and introduce a method to obtain an incremental supervision signal for learning an adaptive revision policy. Experimental results on sequence labelling show that our model has better incremental performance and faster inference speed compared to restart-incremental Transformers, while showing little degradation on full sequences.[1]

1 Introduction

Incrementality is an inseparable aspect of language use. Human speakers can produce utterances based on an incomplete message formed in their minds while simultaneously continuing to refine its content for subsequent speech production (Kempen and Hoenkamp, 1982, 1987). They also comprehend language on (approximately) a word-by-word basis and do not need to wait until the utterance finishes to grasp its meaning (Kamide, 2008).

As observed by Madureira and Schlangen (2020), a natural option for neural network-based incremental processing would be RNNs (Rumelhart et al., 1986), as they have essential properties

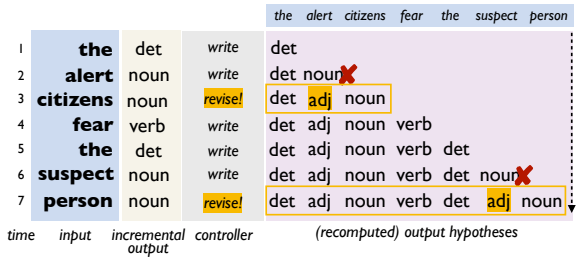

Figure 1: Illustrative example of how a monotonic incremental POS-tagger would not recover from wrong hypotheses. A policy for adaptive revision, here parameterised by a controller, can enable reanalyses to be performed when necessary (here at time steps 3 and 7).

required in incremental scenarios: They keep a recurrent state, are sensitive to the notion of order and are able to accept partial input and produce an output at each time step. Ideally, an incremental processor should also be able to revise its previous incorrect hypotheses based on new input prefixes (Schlangen and Skantze, 2009). However, RNNs are unable to do so as their output is monotonic.

The Transformer architecture (Vaswani et al., 2017) has been the *de facto* standard for many NLP tasks since its inception. Nevertheless, it is not designed for incremental processing as the input sequences are assumed to be complete and processed as a whole. A *restart-incremental* interface (Beuck et al., 2011; Schlangen and Skantze, 2011) can be applied to adapt Transformers for incremental processing (Madureira and Schlangen, 2020), where available input prefixes are recomputed at each time step to produce partial outputs. Such an interface also provides the capability to revise existing outputs through its non-monotonic nature. Although feasible, this method does not scale well for long sequences since the number of required forward passes grows with the sequence length.[2] The revision process is also not effective as it occurs at

[1]Our implementation is publicly available at https://github.com/pkhdipraja/tapir.

[2]Processing a sequence of n tokens once turns into processing n sequences with $\sum_{k=1}^{n} k$ tokens in total.

Findings of the Association for Computational Linguistics: ACL 2023, pages 4173–4197
July 9-14, 2023

every time step, even when it is unnecessary.

Revision is crucial in incremental processing, as it is not always possible for a model to be correct at the first attempt, either because the linguistic input is provided in its inherent piecemeal fashion (as shown in Figure 1) or because of mistakes due to poor approximation. One way to improve the output quality is the delay strategy (Beuck et al., 2011; Baumann et al., 2011), where tokens within a lookahead window are used to disambiguate the currently processed input. However, it can neither fix past hypotheses nor capture long-range influences *e.g.* in garden path sentences.

In this work, we propose the Two-pass model for AdaPtIve Revision (TAPIR), which is capable of adaptive revision, while also being fast in incremental scenarios. This is achieved by using a revision policy to decide whether to `WRITE` (produce a new output) or `REVISE` (refine existing outputs based on new evidence), whose mechanism is described in §3. Learning this policy requires a supervision signal which is usually not present in non-incremental datasets (Köhn, 2018). In §4, we tackle this issue by introducing a method for obtaining action sequences using the Linear Transformer (LT) (Katharopoulos et al., 2020). As silver labels, these action sequences allow us to view policy learning as a supervised problem.

Experiments on four NLU tasks in English, framed as sequence labelling[3], show that, compared to a restart-incremental Transformer encoder, our model is considerably faster for incremental inference with better incremental performance, while being comparable when processing full sequences. Our in-depth analysis inspects TAPIR's incremental behaviour, showing its effectiveness at avoiding ill-timed revisions on correct prefixes.

2 Related Work

There has been increasing interest to explore neural network-based incremental processing. Žilka and Jurčíček (2015) proposed a dialogue state tracker using LSTM (Hochreiter and Schmidhuber, 1997) to incrementally predict each component of the dialogue state. Liu et al. (2019) introduced an incremental anaphora resolution model composed of a memory unit for entity tracking and a recurrent unit as the memory controller. RNNs still fall short on non-incremental metrics due to their strict left-to-right processing. Some works have attempted to address this issue by adapting BiLSTMs or Transformers for incremental processing and applying it on sequence labelling and classification tasks (Madureira and Schlangen, 2020; Kahardipraja et al., 2021) and disfluency detection (Rohanian and Hough, 2021; Chen et al., 2022).

Our revision policy is closely related to the concept of policy in simultaneous translation, which decides whether to wait for another source token (`READ` action) or to emit a target token (`WRITE` action). Simultaneous translation policies can be categorised into fixed and adaptive. An example of a fixed policy is the wait-k policy (Ma et al., 2019), which waits for first k source tokens before alternating between writing and reading a token. An adaptive policy on the other hand, decides to read or write depending on the available context and can be learned by using reinforcement learning techniques (Grissom II et al., 2014; Gu et al., 2017) or applying monotonic attention (Raffel et al., 2017; Chiu and Raffel, 2018; Arivazhagan et al., 2019; Ma et al., 2020).

The memory mechanism is a key component for revision policy learning as it stores representations which, for instance, can be used to ensure that the action is correct (Guo et al., 2022). It also absorbs asynchronies that may arise when each component in an incremental system has different processing speed (Levelt, 1993). The memory can be internal as in RNNs, or external such as memory networks (Weston et al., 2015; Sukhbaatar et al., 2015) and the Neural Turing Machine (Graves et al., 2014).

Revision in incremental systems has been previously explored. In simultaneous spoken language translation, Niehues et al. (2016) proposed a scheme that allows re-translation when an ASR component recognises a new word. Arivazhagan et al. (2020) evaluated streaming translation against re-translation models that translate from scratch for each incoming token and found that re-translation yields a comparable result to streaming systems. Zheng et al. (2020) proposed a decoding method for simultaneous translation that overgenerates target words at each step, which are subsequently revised. One way to achieve revision is by employing a two-pass strategy. Xia et al. (2017) proposed a deliberation network for machine translation, composed of encoder-decoder architecture with an additional second-pass decoder to refine the generated

[3]We do not run experiments on sequence classification, as revisions can trivially be performed by predicting one label at every time step.

target sentence. In dialogue domains, this strategy is also used to improve the contextual coherence and correctness of the response (Li et al., 2019) and to refine the output of retrieval-based dialogue systems (Song et al., 2018; Weston et al., 2018). Furthermore, the two-pass approach is commonly utilised in streaming ASR to improve the initial hypothesis (Sainath et al., 2019; Hu et al., 2020; Wang et al., 2022, *inter alia*).

The aforementioned works shared a common trait, as they used a fixed policy and performed revision either for each incoming input or when the input is already complete. Our approach differs in that our model learns an adaptive policy that results in more timely revisions. Contemporaneous to our work, Kaushal et al. (2023) proposed a cascaded uni- and bidirectional architecture with an additional module to predict when to restart. The module is trained with a supervision signal obtained from comparing the model's prediction against the ground truth. Their approach is effective in reducing the required computational budget.

3 Model

To address the weaknesses of RNN- and Transformer-only architectures for incremental processing (§1), we introduce a Two-pass model for AdaPtIve Revision named TAPIR, which integrates advantages of both models and is based on the deliberation network (Xia et al., 2017). Its architecture, depicted in Figure 2, consists of four components as follows:

1. **Incremental Processor**: a recurrent model that produces an output at each time step and serves as the first-pass model. In this work, we use a standard LSTM network.

2. **Reviser**: a bidirectional model that can revise via recomputation operations (§3.1), also called the second-pass model. We opt for Transformer-based models following Li et al. (2020) as it allows parallel recomputation. The revision process corresponds to the *forward reanalysis hypothesis* (Frazier and Rayner, 1982), where a sentence is processed from the beginning whenever the need for reanalysis is detected.

3. **Memory**: the history of inputs and outputs are stored in the memory. Taking the inspiration from Grave et al. (2017), we use caches, as they are computationally cheap, offering a considerable speed-up in incremental settings.

4. **Controller**: a neural network that parameterises the revision policy. We choose a recurrent controller following Graves et al. (2014), as its internal memory complements the memory module and is also suitable for incremental scenarios. We use a modified LSTMN (Cheng et al., 2016) for this component.

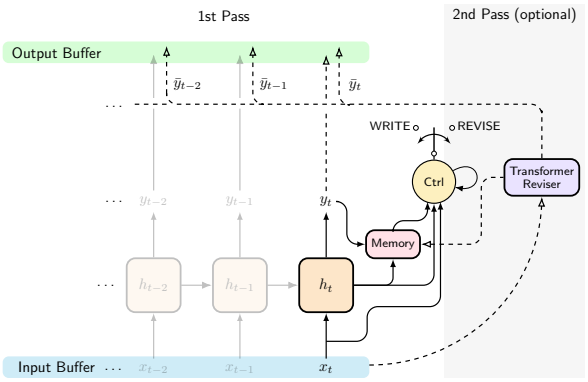

Figure 2: TAPIR computes a candidate output using an RNN at each time step. Then the controller decides whether to WRITE by adding the new output to the output buffer or to take a REVISE action, which can edit the output buffer after observing the effect of the new input on past outputs with the help of the memory.

During incremental inference, TAPIR computes a candidate output y_t for the most recent input x_t as the first pass. Then, based on x_t and the memory, it decides whether to take a WRITE (add y_t to an output buffer) or REVISE (perform a second pass to recompute all existing outputs) action. The action is defined by a revision policy π_θ, which models the effect of new input on past outputs. At each time t, π_θ makes use of processed inputs $x_{\leq t}$ and past outputs $y_{<t}$ to select a suitable action a_t.[4] It is parameterised by the controller hidden state k_t with a non-linear function g:

$$\pi_\theta(a_t|a_{<t}, x_{\leq t}, y_{<t}) \propto g_\theta(k_t) \quad (1)$$

3.1 Revision Policy

In restart-incremental models, revisions can occur as a result of recomputations, which are costly since they happen at every time step, even when no revisions occur. TAPIR revises by selectively deciding when to recompute, which enables it to

[4]The output y_t is excluded as it is not required to determine if a recomputation should occur in our model.

revisit previous outputs at different points in time while reducing the number of recomputations.

Memory Content. The memory in TAPIR contains information pertaining to processed inputs and their corresponding outputs, which is crucial for our approach. This is because it enables our model to perform relational learning between an incoming input and past outputs, using past inputs as an additional cue. Here, we use three caches Γ. Γ^h stores the hidden state h of the incremental processor, representing the current input prefix, Γ^z stores the projected output vector z which represents the output, and Γ^p stores the input-output representation φ, which is computed from h and z. The i-th slot of the caches contains $\gamma_i^h, \gamma_i^z, \gamma_i^p$, all of them computed at the same time step. The representations z and φ are computed as follows:

$$z = \tanh(W_{\tilde{y}}\tilde{y} + b_z) \tag{2}$$

$$\varphi = \tanh(W_{in}h + W_{out}z + b_\varphi) \tag{3}$$

where $\tilde{y}$ is the output logits from the incremental processor. $W_{\tilde{y}}, W_{in}$, and W_{out} are parameters while b_z and b_φ are bias terms. The dimension of z and h is the same. We keep the cache size N small, as we later perform soft attention over Γ^p. The attention computation for large cache sizes is costly and is not suitable for incremental scenarios. Due to this limitation, the oldest cache element is discarded when the cache is full and new partial input arrives.

Modelling Actions. To model possible changes in past outputs as an effect of a new input, we use an LSTMN controller due to its ability to induce relations among tokens. It computes the relation between h_t and each cache element γ_i^p via an attention mechanism:

$$U = W_c\gamma_i^p + W_h h_t + W_{\tilde{k}}\tilde{k}_{t-1} + b_u \tag{4}$$

$$s_i^t = \text{softmax}(v^\top \tanh(U)) \tag{5}$$

which yields a probability distribution over Γ^p. $\tilde{k}_{t-1}$ is the previous summary vector of the controller hidden state. $W_c, W_h, W_{\tilde{k}}$, and v are parameters and b_u is a bias term. We can then compute adaptive summary vectors $\tilde{k}_t$ and $\tilde{c}_t$ as a weighted sum of the cache Γ^p and the controller memory tape C_{t-1}:

$$\begin{bmatrix}\tilde{k}_t \\ \tilde{c}_t\end{bmatrix} = \sum_{i=1}^{N} s_i^t \cdot \begin{bmatrix}\gamma_i^p \\ c_{i+\max(0,t-N-1)}\end{bmatrix} \tag{6}$$

where $c_{i+\max(0,t-N-1)}$ is the controller memory cell for the corresponding cache element γ_i^p. The attention can be partially viewed as local (Luong et al., 2015), since older cache elements are incorporated through $\tilde{k}_{t-1}$. These summary vectors are used to compute the recurrent update as follows:

$$\begin{bmatrix}i_t \\ f_t \\ o_t \\ \hat{c}_t\end{bmatrix} = \begin{bmatrix}\sigma \\ \sigma \\ \sigma \\ \tanh\end{bmatrix} W \cdot [\tilde{k}_t, x_t] \tag{7}$$

$$c_t = f_t \odot \tilde{c}_t + i_t \odot \hat{c}_t \tag{8}$$

$$k_t = o_t \odot \tanh(c_t) \tag{9}$$

Lastly, k_t is used by the revision policy to compute the action a_t:

$$\pi_\theta(a_t|a_{<t}, x_{\leq t}, y_{<t}) = \sigma(\theta^\top k_t + b_k) \tag{10}$$

$$a_t = \begin{cases}\texttt{REVISE}, & \text{if } \sigma(\theta^\top k_t + b_k) \geq \tau \\ \texttt{WRITE}, & \text{otherwise}\end{cases} \tag{11}$$

where θ is a parameter vector, b_k is the bias, and $\tau \in [0, 1]$ is a decision threshold. According to equation (11), a REVISE action is selected only if the policy value is greater than or equal to τ; otherwise, a WRITE action is chosen. This threshold can be adjusted to encourage or discourage the recomputation frequency without the need to retrain the policy. Our model is equal to an RNN when $\tau = 1$ (never recompute), and becomes a restart-incremental Transformer when $\tau = 0$ (always recompute).

3.2 Incremental Inference Mechanism

Using the policy, TAPIR predicts when to perform a recomputation. Assume that an input token x_t is fed to the RNN component to obtain y_t. The controller then reads x_t, h_t, and Γ^p to compute a_t. If a REVISE action is emitted, the input buffer (containing all available inputs so far) will be passed to the reviser to yield the recomputed outputs. When this happens, both z and φ stored in the caches also need to be updated to reflect the effect of the recomputation. The recomputation of past z and φ will occur simultaneously with the computation of z and φ for the current time step to update Γ^z and Γ^p using the recomputed outputs. If a WRITE action is emitted, we take y_t to be the current output and continue to process the next token. The content of Γ^z and Γ^p are also updated for the current step. The cache Γ^h is always updated regardless of which action the policy takes. See algorithm in the Appendix.

Let us use Figure 1 and $\tau = 0.5$ as a constructed example. At $t = 1$, the incremental processor consumes the token *the*, updates its hidden state and predicts the POS-tag `det`. The controller predicts that the probability for recomputation is *e.g.* 0.3. Since it is lower than τ, `det` gets written to the output buffer, the memory is updated and the current step is finished. A similar decision happens at $t = 2$ and *alert* is classified as `noun`. At $t = 3$, however, the controller predicts that a `REVISE` action should occur after the input *citizens*. That triggers the reviser, which takes *the alert citizens* as input and returns `det adj noun`. The output buffer gets overwritten with this new hypothesis and the caches are recomputed to accommodate the new state. This dynamics continues until the end of the sentence.

3.3 Training

Jointly training all components of such a two-pass model from scratch can be unstable (Sainath et al., 2019), so we opt for a two-step training process:

1. Train only the reviser using cross entropy loss.

2. Train the incremental processor and the controller together with a combined loss:

$$\mathcal{L} = CE(y^{\text{gold}}, y) + BCE(a^{\text{LT}}, a) \quad (12)$$

where y^{gold} is the expected output and a^{LT} is the expected action.

4 Supervision Signal for Revision

During incremental sentence comprehension, a revision or reanalysis occurs when disambiguating material rules out the current sentence interpretation. In Figure 1, `noun` is a valid label for *suspect* at $t = 6$, but *person* at $t = 7$ rules that analysis out, forcing a reanalysis to `adj` instead.

Training TAPIR's controller requires a sequence of `WRITE`/`REVISE` actions expressed as the supervision signal a^{LT} in equation (12), capturing when revision happens. This signal then allows us to frame the policy learning as a supervised learning task (as in the work of Zheng et al. (2019)).

If we have the sequence of output prefix hypotheses at each step, as shown in Figure 1, we know that the steps when revisions have occurred are $t = \{3, 7\}$. We can then construct the sequence of actions we need. The first action is always `WRITE` as there is no past output to revise at this step. For $t > 1$, the action can be determined by comparing the partial outputs at time step t (excluding y_t) against the partial outputs at time step $t - 1$. If no edits occur, then the partial outputs after processing x_t should not change, and a `WRITE` action is appended to the sequence. If any edits occur, we append a `REVISE` action instead.

Intermediate human judgements about when to revise are not available, so we need to retrieve that from a model. It is possible obtain this information from a restart-incremental Transformer, by comparing how the prefix at t differs from prefix at $t - 1$. However, as shown by Kahardipraja et al. (2021), the signal captured using this approach may lack incremental quality due to the missing recurrence mechanism. Using a recurrent model is advisable here, as it can capture order and hierarchical structure in sentences, which is apparently hard for Transformers (Tran et al., 2018; Hahn, 2020; Sun and Lu, 2022). But it is difficult to retrieve this signal using vanilla RNNs because its recurrence only allows a unidirectional information flow, which prevents a backward update of past outputs.

Therefore, we opt for the Linear Transformer (LT) (Katharopoulos et al., 2020), which can be viewed both as a Transformer and as an RNN.[5] To generate the action sequences, we first train the action generator LT with causal mask to mimic an RNN training. Afterwards, it is deployed under restart-incrementality on the same set used for training with the mask removed. We collect the sequence of partial prefixes for all sentences and use it to derive the action sequences.

5 Experiments

Datasets. We evaluate TAPIR on four tasks in English, for NLU and task-oriented dialogue, using seven sequence labelling datasets:

- ▷ Slot Filling: SNIPS (Coucke et al., 2018); Alarm, Reminder & Weather (Schuster et al., 2019) and MIT Movie (Liu et al., 2013).
- ▷ PoS Tagging: CoNLL-2003 (Tjong Kim Sang and De Meulder, 2003) and UD-EWT (Silveira et al., 2014).
- ▷ Named-Entity Recognition (CoNLL-2003).
- ▷ Chunking (CoNLL-2003).

[5] For a detailed proof of why LT is more suitable, see the Appendix.

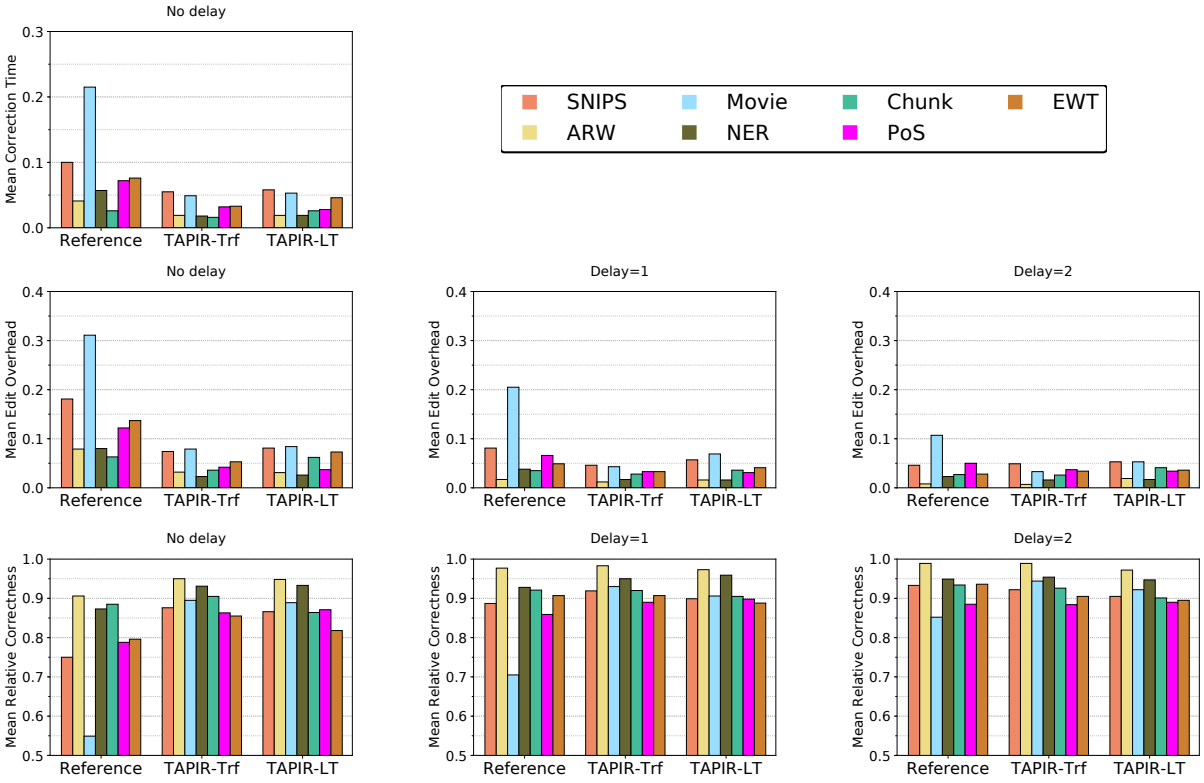

Figure 3: Incremental evaluation of the models on test sets. Edit Overhead, Correction Time Score and Relative Correctness $\in [0, 1]$. Lower is better for EO and CT, while higher is better for RC. TAPIR is better compared to the reference model for the non-delayed case (output prefixes are often correct and stable). The delay strategy of one lookahead token is beneficial.

Table 1 shows the distribution of generated actions in the final training set for each task. Further details regarding the datasets and generated action sequences are available in the Appendix.

Tasks	WRITE	REVISE
SNIPS	0.777	0.223
ARW	0.811	0.189
Movie	0.765	0.235
NER	0.895	0.105
Chunk	0.687	0.313
PoS	0.769	0.231
EWT	0.712	0.288

Table 1: Distribution of generated actions (train+val).

Evaluation. An ideal incremental model deployed in real-time settings should (i) exhibit good incremental behaviour, *i.e.* produce correct and stable partial hypotheses and timely recover from its mistakes; (ii) be efficient for inference by delivering responses without wasting computational resources; and (iii) not come with the cost of a negative impact on the non-incremental performance, *i.e.* produce correct final outputs. Achieving all at the same time may be hard, so trade-offs can be necessary.

We evaluate TAPIR on these three relevant dimensions. For (i), we use similarity and diachronic metrics[6] proposed by Baumann et al. (2011) and adapted in Madureira and Schlangen (2020): *edit overhead* (EO, the proportion of unnecessary edits over all edits), *correction time score* (CT, the average proportion of time steps required for an output increment to settle down), and *relative correctness* (RC, the proportion of output prefixes that match with the final output). Aspect (ii) is analysed by benchmarking the incremental inference speed. For (iii), we use the F1 score adapted for the IOB sequence labelling scheme, except for PoS tagging, which is evaluated by measuring accuracy.

Rather than trying to beat the state-of-the art results, we focus on analysing the incremental abilities of models whose performances are high enough for our purposes. As a reference model, we use a Transformer encoder applied in a restart-incremental fashion, which implicitly performs revision at every step. We follow Baumann et al. (2011) and Madureira and Schlangen (2020) by evaluating partial outputs with respect to the final

[6]This metric can also be used for incremental evaluation involving frame semantics. See Atterer et al. (2009) for details.

output, to separate between incremental and non-incremental performance.

Delay strategy. To inspect the effect of right context on the model's performance, we use the delay strategy (Baumann et al., 2011) with a lookahead window of size 1 and 2, computing a delayed version of EO and RC (Madureira and Schlangen, 2020). The output for the reference model is delayed only during inference, as in Madureira and Schlangen (2020). For TAPIR, the same treatment would not be possible as it contains an RNN that must be able to recognise the output delay. Thus, we follow the approach of Turek et al. (2020): During training and inference, the label for input x_t is expected at time step $t + d$, where d is the delay.

Implementation. For the reviser component, we choose Transformer (Trf) and Linear Transformer (LT) encoders trained with full attention.[7] The reference model is trained with cross entropy loss similar to the reviser. All models are trained with the AdamW optimiser (Loshchilov and Hutter, 2019). We use 300-D GloVe embeddings (Pennington et al., 2014), which, for the reference model and the reviser, are passed through an additional linear projection layer. The probability threshold τ is set to 0.5. We report results for a single run with the best hyperparameter configuration. See Appendix for details about the set-up and experiments.

6 Results and Analysis

Incremental. Figure 3 depicts the incremental evaluation results. For the no-delay case, TAPIR performs better compared to the reference model. We also observe that the delay strategy helps improve the metrics. It improves the results for TAPIR, in general, but a longer delay does not always yield a better incremental performance. We suspect this happens for two possible reasons: First, if we consider the case where the delay is 1, TAPIR has already achieved relatively low EO (< 0.1) and high RC (> 0.85). This, combined with its non-monotonic behaviour, might make it harder to further improve on both incremental metrics, even if a longer delay is allowed. Second, a longer delay means that our model needs to wait longer before producing an output. In the meantime, it still has to process incoming tokens, which might cause some difficulty in learning the relation between the input and its corresponding delayed output. As a consequence, we have mixed results when comparing EO and RC for the delayed version of the reference model and TAPIR. Their differences are, however, very small. TAPIR achieves low EO and CT score, which indicates that the partial output is stable and settles down quickly. RC is also high, which shows that, most of the time, the partial outputs are correct prefixes of the final, non-incremental output and would be useful for downstream processing.

Benchmark. Table 2 shows that TAPIR is considerably faster compared to the reference model in incremental settings, as it offers, on average, $\sim$4.5$\times$ speed-up in terms of sequences per second.[8]

Tasks	Ref.	TAPIR-Trf	TAPIR-LT
SNIPS	1.103	4.958 (4.50×)	**8.983 (8.15×)**
ARW	2.339	**8.734 (3.73×)**	5.959 (2.55×)
Movie	0.927	**3.520 (3.80×)**	3.432 (3.70×)
NER	0.675	4.465 (6.62×)	**4.502 (6.67×)**
Chunk	0.688	**2.714 (3.95×)**	1.912 (2.78×)
PoS	0.672	4.111 (6.12×)	**7.400 (11.01×)**
EWT	0.819	**3.659 (4.47×)**	3.122 (3.81×)
Average	1.032	4.594 (4.45×)	**5.044 (4.89×)**

Table 2: Comparison of incremental inference speed on test sets. TAPIR is $\sim$4.5$\times$ faster compared to the reference model. All results are in sentences/sec.

Non-Incremental. The performance of the restart-incremental reference model and our model on full sentences is shown in Table 3. The results of TAPIR, in particular with the Transformer reviser (TAPIR-Trf), are roughly comparable to the reference model, with only modest differences (0.96% – 4.12%). TAPIR-Trf performs slightly better than TAPIR-LT. This is possibly due to the approximation of softmax attention in LT, which leads to degradation in the output quality. Furthermore, we see that delay of 1 or 2 tokens for TAPIR is generally beneficial.[9] Note that we do not force a `REVISE` action at the final time step to examine the effect of the learned policy on TAPIR's performance, although that would be a strategy to achieve the same non-incremental performance as the reference model.

[7] We previously tried to train both revisers with causal mask to make revision more robust as it can occur before the input is complete, however our preliminary results show that training without mask yields better results.

[8] We use the same models as in Table 3 and Figure 3 for benchmarking, as the policy affects the inference speed.

[9] As TAPIR primarily processes sequences left-to-right and only recomputes upon emission of a `REVISE` action, the delay strategy helps to provide more information when making a decision, unlike the reference model that already has access to full sequences.

Tasks	Ref.	TAPIR-Trf -	TAPIR-Trf D1	TAPIR-Trf D2	TAPIR-LT -	TAPIR-LT D1	TAPIR-LT D2
SNIPS	**91.05**	88.57	90.47	89.45	85.95	88.07	87.28
ARW	**95.63**	93.35	95.17	95.15	92.84	93.65	94.50
Movie	**83.98**	82.85	83.26	82.95	81.40	83.16	82.21
NER	**78.25**	74.13	76.85	78.04	73.12	73.79	75.75
Chunk	**88.35**	86.85	87.48	87.52	85.03	86.43	85.79
PoS	**92.28**	91.32	91.35	91.49	90.90	90.83	90.65
EWT	**92.14**	90.84	92.00	91.95	90.20	91.34	90.93

Table 3: Non-incremental performance of the models on test sets (first group is F1, second group is accuracy). D = delay. The performance of TAPIR is roughly comparable to the reference model.

6.1 Detailed Analysis

In the next paragraphs, we assess TAPIR-Trf on aspects beyond the basic evaluation metrics.

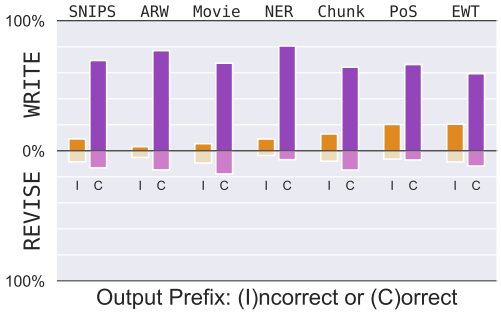

Figure 4: Distribution of actions and output prefixes by dataset. Most of the actions are WRITE and most of the partial prefixes which are correct do not get unnecessarily revised. Incorrect prefixes cannot always be immediately detected, as expected. Part of the REVISE actions are dispensable, but in a much lower frequency than in the restart-incremental paradigm.

Policy Effectiveness. Figure 4 shows the distributions of actions and states of the output prefixes. Here, a prefix is considered correct if all its labels match the final output, and incorrect otherwise. We start by noticing that most of the actions are WRITE, and among them, very few occur when the prefix is incorrect. TAPIR is thus good at recognising states where recomputation is not required, supporting its speed advantage. A good model should avoid revising prefixes that are already correct. We see that, for all datasets, the vast majority of the correct prefixes indeed do not get revised. A utopian model would not make mistakes (and thus never need to revise) or immediately revise incorrect prefixes. In reality, this cannot be achieved, given the incremental nature of language and the long-distance dependencies. As a result, incorrect prefixes are expected

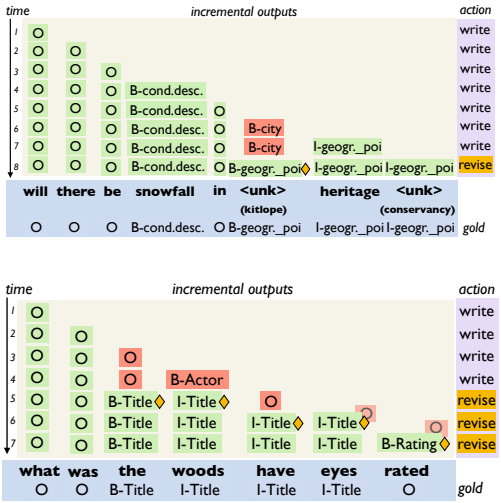

Figure 5: Examples of incremental inference (from SNIPS and Movie) for TAPIR-Trf. Edited labels are marked by a diamond symbol, with the immediate past output at the top right corner for right-frontier edits. Red labels are incorrect with respect to the final output.

to have a mixed distribution between actions, as the model needs to wait for the edit-triggering input, and our results corroborate that. Finally, among the REVISE actions (*i.e.* the lighter bars in the bottom area), there is still a considerable relative number of unnecessary revisions occurring for correct prefixes. We see room for further refinement of the policy in that sense, but, in absolute numbers, the occurrence of recomputations is much lower than in the restart-incrementality paradigm, where all steps require a recomputation.

Qualitative analysis. Figure 5 shows two examples of how TAPIR behaves in incremental slot filling (more examples in the Appendix), showing that it performs critical revisions that would not be possible with a monotonic model.

At the top, the model must produce labels for unknown tokens, which is harder to perform correctly. The first UNK token is initially interpreted as a city at $t = 6$, which is probably deemed as correct considering the available left context. The controller agrees with this, producing a WRITE action. However, when *heritage* and the second UNK token have been consumed at $t = 8$, the incremental processor labels them as parts of a geographic point of interest. The controller is able to notice the output inconsistency as I-geographic_poi should be preceded by B-geographic_poi (following the IOB scheme) and emits a REVISE action. As a result, the label B-city is correctly replaced.

In the second example, TAPIR produces interest-

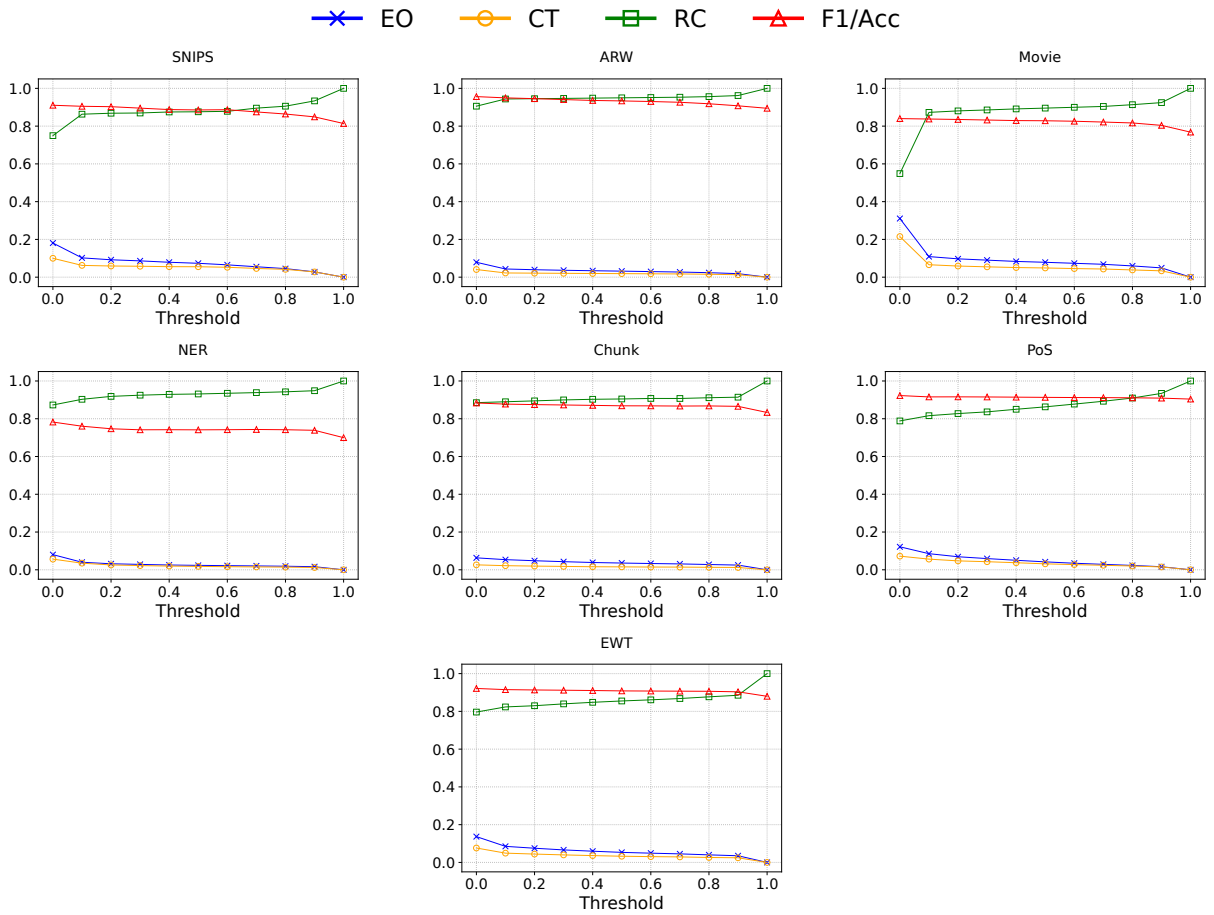

Figure 6: Effect of the probability threshold τ on incremental and non-incremental metrics, using TAPIR-Trf. Increasing τ leads to improvement of incremental metrics at the cost of non-incremental performance.

ing interpretations. It initially considers *woods* to be an actor name at $t = 4$. When it reads *have*, the reanalysis triggered by the controller interprets *woods* as a part of a title, *the woods*. The model revises its hypothesis again at $t = 6$, and decides that the complete title should be *the woods have eyes*. It still makes a mistake at the last time step, opting for a (wrong) revision of `O` to `B-RATING` for *rated* when it should be unnecessary.

Effect of Threshold. Figure 6 portrays the effect of the probability threshold τ on incremental and non-incremental metrics. As τ increases, the incremental performance improves while the non-incremental performance deteriorates. This happens as higher τ discourages recomputation and makes the model closer to an RNN. In return, it is harder for the model to revisit its past decisions.

7 Conclusion

We proposed TAPIR, a two-pass model capable of performing adaptive revision in incremental scenarios *e.g.* for dialogue and interactive systems. We also demonstrated that it is possible to obtain an incremental supervision signal using the Linear Transformer (LT), in the form of `WRITE`/`REVISE` action sequences, to guide the policy learning for adaptive revision. Results on sequence labelling tasks showed that TAPIR has a better incremental performance than a restart-incremental Transformer, in general, while being roughly comparable to it on full sentences. The delay strategy helps to improve incremental and non-incremental metrics, although a longer delay does not always yield better results.

The ability to revise adaptively provides our model with substantial advantages over using RNNs or restart-incremental Transformers. It can fix incorrect past outputs after observing incoming inputs, which is not possible for RNNs. Looking from the aspect of efficiency, our model is also better compared to restart-incremental Transformers as the recomputation is only performed when the need for it is detected. TAPIR is consequently faster in terms of inference speed.

Limitations

In this section, we discuss some of the known limitations of our set-up, data and models.

To handle unknown words in the test sets, we replace them by a special UNK token which is also used to mask some tokens in the training set. The UNK token provides little information regarding the actual input and TAPIR might be unable to fully utilise the token to refine its interpretation of the past output. This has a direct influence in the incremental metrics, as the model can exploit this property by using UNK token as a cue to emit the REVISE action. This strategy also introduces the extra hyperparameter of what proportion of tokens to mask.

We put effort into achieving a diverse selection of datasets in various tasks, but our analysis is limited to English. We are reporting results on the datasets for which the non-incremental versions of the model could achieve a performance high enough to allow a meaningful evaluation of their incremental performance. Tuning is still required to extend the analysis to other datasets.

Related to these two issues, we decided to use tokens as the incremental unit for processing. We follow the tokenization given by the sequence labelling datasets we use. Extending the analysis for other languages requires thus a good tokenizer, and annotated data, which may not exist. We may also inherit limitations from the datasets that we use. Although we do not include an in-depth analysis of the datasets, as our focus is on the model and not on solving the tasks themselves, they are widely used by the community and details are available in their corresponding publications.

The method we propose to retrieve the action sequences depends on the chosen model, and the grounding of the action sequences in the actual prefix outputs have a direct influence in training the controller. Therefore, the decisions made by TAPIR rely on the quality of the underlying generated action sequences. In order to ensure that the internal representations of the action generator LT do not depend on right context, we had to restrict ourselves to a single layer variation of this model when generating the sequence of actions. It is possible that with more layers its behaviour would be different, but that would invalidate the assumptions needed for an incremental processor.

When it comes to the TAPIR architecture, the attention scores for the controller are computed independently of temporal order and we do not explicitly model relation between cache elements. The limited cache size also means that some past information has to be discarded to accommodate incoming inputs. Although we have made efforts to incorporate them through the summary vector, this might be not ideal due to information bottleneck.

Ethics Statement

We do not see any immediate ethical issues arising from this work, beyond those inherent to NLP which are under discussion by the community.

Acknowledgements

We thank the anonymous reviewers for their valuable and insightful comments and suggestions. This work is partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project ID 423217434 (Schlangen).

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2623–2631, New York, NY, USA. Association for Computing Machinery.

Naveen Arivazhagan, Colin Cherry, Wolfgang Macherey, Chung-Cheng Chiu, Semih Yavuz, Ruoming Pang, Wei Li, and Colin Raffel. 2019. Monotonic infinite lookback attention for simultaneous machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1313–1323, Florence, Italy. Association for Computational Linguistics.

Naveen Arivazhagan, Colin Cherry, Wolfgang Macherey, and George Foster. 2020. Re-translation versus streaming for simultaneous translation. In *Proceedings of the 17th International Conference on Spoken Language Translation*, pages 220–227, Online. Association for Computational Linguistics.

Michaela Atterer, Timo Baumann, and David Schlangen. 2009. No sooner said than done? testing incrementality of semantic interpretations of spontaneous speech. In *INTERSPEECH 2009, 10th Annual Conference of the International Speech Communication Association, Brighton, United Kingdom, September 6-10, 2009*, pages 1855–1858. ISCA.

Timo Baumann, Okko Buß, and David Schlangen. 2011. Evaluation and optimisation of incremental processors. *Dialogue & Discourse*, 2(1):113–141. Special Issue on Incremental Processing in Dialogue.

Niels Beuck, Arne Köhn, and Wolfgang Menzel. 2011. Decision strategies for incremental POS tagging. In *Proceedings of the 18th Nordic Conference of Computational Linguistics (NODALIDA 2011)*, pages 26–33, Riga, Latvia. Northern European Association for Language Technology (NEALT).

Angelica Chen, Vicky Zayats, Daniel Walker, and Dirk Padfield. 2022. Teaching BERT to wait: Balancing accuracy and latency for streaming disfluency detection. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 827–838, Seattle, United States. Association for Computational Linguistics.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. 2018. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Jianpeng Cheng, Li Dong, and Mirella Lapata. 2016. Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 551–561, Austin, Texas. Association for Computational Linguistics.

Chung-Cheng Chiu and Colin Raffel. 2018. Monotonic chunkwise attention. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2016. Fast and accurate deep network learning by exponential linear units (elus). In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, Maël Primet, and Joseph Dureau. 2018. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. *arXiv preprint*, arXiv:1805.10190.

Haihong E, Peiqing Niu, Zhongfu Chen, and Meina Song. 2019. A novel bi-directional interrelated model for joint intent detection and slot filling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5467–5471, Florence, Italy. Association for Computational Linguistics.

Lyn Frazier and Keith Rayner. 1982. Making and correcting errors during sentence comprehension: Eye movements in the analysis of structurally ambiguous sentences. *Cognitive Psychology*, 14(2):178–210.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

Edouard Grave, Armand Joulin, and Nicolas Usunier. 2017. Improving neural language models with a continuous cache. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *CoRR*, abs/1410.5401.

Alvin Grissom II, He He, Jordan Boyd-Graber, John Morgan, and Hal Daumé III. 2014. Don't until the final verb wait: Reinforcement learning for simultaneous machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1342–1352, Doha, Qatar. Association for Computational Linguistics.

Jiatao Gu, Graham Neubig, Kyunghyun Cho, and Victor O.K. Li. 2017. Learning to translate in real-time with neural machine translation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1053–1062, Valencia, Spain. Association for Computational Linguistics.

Shoutao Guo, Shaolei Zhang, and Yang Feng. 2022. Turning fixed to adaptive: Integrating post-evaluation into simultaneous machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2264–2278, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Michael Hahn. 2020. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Ke Hu, Tara N. Sainath, Ruoming Pang, and Rohit Prabhavalkar. 2020. Deliberation model based two-pass end-to-end speech recognition. In *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*, pages 7799–7803. IEEE.

Patrick Kahardipraja, Brielen Madureira, and David Schlangen. 2021. Towards incremental transformers: An empirical analysis of transformer models for incremental NLU. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1178–1189, Online and Punta Cana,

Dominican Republic. Association for Computational Linguistics.

Yuki Kamide. 2008. Anticipatory processes in sentence processing. *Language and Linguistics Compass*, 2(4):647–670.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5156–5165. PMLR.

Ayush Kaushal, Aditya Gupta, Shyam Upadhyay, and Manaal Faruqui. 2023. Efficient encoders for streaming sequence tagging. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 418–429, Dubrovnik, Croatia. Association for Computational Linguistics.

Gerard Kempen and Edward Hoenkamp. 1982. Incremental sentence generation: Implications for the structure of a syntactic processor. In *Coling 1982: Proceedings of the Ninth International Conference on Computational Linguistics*.

Gerard Kempen and Edward Hoenkamp. 1987. An incremental procedural grammar for sentence formulation. *Cognitive Science*, 11(2):201–258.

Arne Köhn. 2018. Incremental natural language processing: Challenges, strategies, and evaluation. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2990–3003, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Willem J. M. Levelt. 1993. *Speaking: From Intention to Articulation*. The MIT Press.

Wei Li, James Qin, Chung-Cheng Chiu, Ruoming Pang, and Yanzhang He. 2020. Parallel rescoring with transformer for streaming on-device speech recognition. In *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association, Virtual Event, Shanghai, China, 25-29 October 2020*, pages 2122–2126. ISCA.

Zekang Li, Cheng Niu, Fandong Meng, Yang Feng, Qian Li, and Jie Zhou. 2019. Incremental transformer with deliberation decoder for document grounded conversations. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 12–21, Florence, Italy. Association for Computational Linguistics.

Fei Liu, Luke Zettlemoyer, and Jacob Eisenstein. 2019. The referential reader: A recurrent entity network for anaphora resolution. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5918–5925, Florence, Italy. Association for Computational Linguistics.

Jingjing Liu, Panupong Pasupat, Scott Cyphers, and James R. Glass. 2013. Asgard: A portable architecture for multilingual dialogue systems. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 8386–8390. IEEE.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.

Mingbo Ma, Liang Huang, Hao Xiong, Renjie Zheng, Kaibo Liu, Baigong Zheng, Chuanqiang Zhang, Zhongjun He, Hairong Liu, Xing Li, Hua Wu, and Haifeng Wang. 2019. STACL: Simultaneous translation with implicit anticipation and controllable latency using prefix-to-prefix framework. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3025–3036, Florence, Italy. Association for Computational Linguistics.

Xutai Ma, Juan Miguel Pino, James Cross, Liezl Puzon, and Jiatao Gu. 2020. Monotonic multihead attention. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Matthew MacKay, Paul Vicol, Jimmy Ba, and Roger B Grosse. 2018. Reversible recurrent neural networks. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Brielen Madureira and David Schlangen. 2020. Incremental processing in the age of non-incremental encoders: An empirical assessment of bidirectional models for incremental NLU. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 357–374, Online. Association for Computational Linguistics.

Jan Niehues, Thai Son Nguyen, Eunah Cho, Thanh-Le Ha, Kevin Kilgour, Markus Müller, Matthias Sperber, Sebastian Stüker, and Alex Waibel. 2016. Dynamic transcription for low-latency speech translation. In *Interspeech 2016, 17th Annual Conference of the International Speech Communication Association, San Francisco, CA, USA, September 8-12, 2016*, pages 2513–2517. ISCA.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Colin Raffel, Minh-Thang Luong, Peter J. Liu, Ron J. Weiss, and Douglas Eck. 2017. Online and linear-time attention by enforcing monotonic alignments. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2837–2846. PMLR.

Morteza Rohanian and Julian Hough. 2021. Best of both worlds: Making high accuracy non-incremental transformer-based disfluency detection incremental. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3693–3703, Online. Association for Computational Linguistics.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature*, 323:533–536.

Tara N. Sainath, Ruoming Pang, David Rybach, Yanzhang He, Rohit Prabhavalkar, Wei Li, Mirkó Visontai, Qiao Liang, Trevor Strohman, Yonghui Wu, Ian McGraw, and Chung-Cheng Chiu. 2019. Two-pass end-to-end speech recognition. In *Interspeech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, 15-19 September 2019*, pages 2773–2777. ISCA.

David Schlangen and Gabriel Skantze. 2009. A general, abstract model of incremental dialogue processing. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 710–718, Athens, Greece. Association for Computational Linguistics.

David Schlangen and Gabriel Skantze. 2011. A general, abstract model of incremental dialogue processing. *Dialogue & Discourse*, 2(1):83–111. Special Issue on Incremental Processing in Dialogue.

Sebastian Schuster, Sonal Gupta, Rushin Shah, and Mike Lewis. 2019. Cross-lingual transfer learning for multilingual task oriented dialog. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3795–3805, Minneapolis, Minnesota. Association for Computational Linguistics.

Natalia Silveira, Timothy Dozat, Marie-Catherine de Marneffe, Samuel Bowman, Miriam Connor, John Bauer, and Chris Manning. 2014. A gold standard dependency corpus for English. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 2897–2904, Reykjavik, Iceland. European Language Resources Association (ELRA).

Yiping Song, Cheng-Te Li, Jian-Yun Nie, Ming Zhang, Dongyan Zhao, and Rui Yan. 2018. An ensemble of retrieval-based and generation-based human-computer conversation systems. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4382–4388. International Joint Conferences on Artificial Intelligence Organization.

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. End-to-end memory networks. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Xiaobing Sun and Wei Lu. 2022. Implicit n-grams induced by recurrence. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1624–1639, Seattle, United States. Association for Computational Linguistics.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Ke Tran, Arianna Bisazza, and Christof Monz. 2018. The importance of being recurrent for modeling hierarchical structure. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4731–4736, Brussels, Belgium. Association for Computational Linguistics.

Javier Turek, Shailee Jain, Vy Vo, Mihai Capotă, Alexander Huth, and Theodore Willke. 2020. Approximating stacked and bidirectional recurrent architectures with the delayed recurrent neural network. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9648–9658. PMLR.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Weiran Wang, Ke Hu, and Tara N. Sainath. 2022. Deliberation of streaming rnn-transducer by non-autoregressive decoding. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2022, Virtual and Singapore, 23-27 May 2022*, pages 7452–7456. IEEE.

Jason Weston, Sumit Chopra, and Antoine Bordes. 2015. Memory networks. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Jason Weston, Emily Dinan, and Alexander Miller. 2018. Retrieve and refine: Improved sequence generation models for dialogue. In *Proceedings of the 2018 EMNLP Workshop SCAI: The 2nd International Workshop on Search-Oriented Conversational*

AI, pages 87–92, Brussels, Belgium. Association for Computational Linguistics.

Yingce Xia, Fei Tian, Lijun Wu, Jianxin Lin, Tao Qin, Nenghai Yu, and Tie-Yan Liu. 2017. Deliberation networks: Sequence generation beyond one-pass decoding. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Baigong Zheng, Renjie Zheng, Mingbo Ma, and Liang Huang. 2019. Simpler and faster learning of adaptive policies for simultaneous translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1349–1354, Hong Kong, China. Association for Computational Linguistics.

Renjie Zheng, Mingbo Ma, Baigong Zheng, Kaibo Liu, and Liang Huang. 2020. Opportunistic decoding with timely correction for simultaneous translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 437–442, Online. Association for Computational Linguistics.

Lukáš Žilka and Filip Jurčíček. 2015. Lectrack: Incremental dialog state tracking with long short-term memory networks. In *Text, Speech, and Dialogue*, pages 174–182, Cham. Springer International Publishing.

A Appendix

In this section, we provide information regarding the hyperparameters, implementation, and additional details that are needed to reproduce this work (Table 4 - 14). We also present supplementary materials to accompany the main text (Proof for §4, Algorithm 1, Figure 7 - 8).

For all of our experiments, the seed is set to 42119392. We re-implement the Transformer and the LSTMN used in this work, while for the Linear Transformer (LT), we use the official implementation.[10] Further information regarding dependencies and versions are available in the repository.

Datasets

Tables 6 and 7 summarise the datasets. For SNIPS, we use the preprocessed data and splits provided by E et al. (2019). As the MIT Movie dataset does not have an official validation set, we randomly select 10% of the training data as the validation set. We also remove sentences longer than 200 words. While we use the validation set to tune the hyperparameters of our models, the results on test sets are obtained by using models that are trained on the combination of training and validation sets.

Action Sequence Generation

For the action sequence generation, we train a single-layer LT for 20 epochs with linear learning rate warm-up over the first 5 epochs. We use AdamW optimiser (Loshchilov and Hutter, 2019) with $\beta_1 = 0.9$ and $\beta_2 = 0.98$. Xavier initialisation (Glorot and Bengio, 2010) is applied to all parameters. The learning rate is set to $1e^{-4}$, with gradient clipping of 1, dropout of 0.1, and batch size of 128. We set the FFNN dimension to 2048 and self-attention dimension to 512, with 8 attention heads. The same hyperparameters are used for all datasets. Action sequences for training the final models are obtained using single-layer LTs that are trained on the combination of training and validation sets.

Implementation and training details

Our reference model and TAPIR are trained for 50 epochs with dropout of 0.1 and early stopping with patience of 10. For AdamW, we use $\beta_1 = 0.9$ and $\beta_2 = 0.98$. We also apply Xavier initialisation to all parameters. To train the reference model and the reviser, we use linear learning rate warmup over the first 5 epochs. The learning rate is decayed by 0.5 after 30, 40, and 45 epochs for all models. The number of attention heads for Transformer and LT encoders is set to 8, where each head has the dimension of $d_{\text{model}}/8$ and d_{model} is the self-attention dimension. The embedding projection layer is of size d_{model}. For OOV words, we follow Žilka and Jurčíček (2015) by randomly replacing tokens with an UNK token during training with a probability that we set to 0.02, and then use this token whenever we encounter unknown words during inference. Hyperparameter search is performed using Optuna (Akiba et al., 2019) by maximising the corresponding non-incremental metric on the validation set. We limit the hyperparameter search trials to 25 for all of our experiments. Different from the two-pass model in Sainath et al. (2019), during training we do not take the trained reviser in step (1), freeze its weights, and use it for training step (2). This is because when recomputation occurs, we use output logits from the reviser to recompute z and φ, but this would mean that the error from the previous z and φ cannot be backpropagated. We also experimented using unit logits $(\tilde{y}/\|\tilde{y}\|)$ to compute z, as the logits value from the incremental processor and the reviser might differ in magnitude, but using raw logits proved to be more effective. All the experiments were conducted on a GeForce GTX 1080 Ti and took ∼2 weeks to complete.

Overview of the Linear Transformer

The Linear Transformer (LT) (Katharopoulos et al., 2020) uses kernel-based formulation and associative property of matrix products to approximate the softmax attention in conventional Transformers, which is a special case of self-attention. In LT, the self-attention for the i-th position is expressed as:

$$\text{Att}_i(Q, K, V) = \frac{\phi(Q_i)^\top S_p}{\phi(Q_i)^\top Z_p} \tag{13}$$

$$S_p = \sum_{j=1}^{p} \phi(K_j) V_j^\top ; Z_p = \sum_{j=1}^{p} \phi(K_j) \tag{14}$$

For unmasked attention with a sequence length of N, $p = N$ whereas $p = i$ for causal attention. The feature map ϕ is an exponential linear unit (elu) (Clevert et al., 2016), specifically $\phi(x) = \text{elu}(x) + 1$. LT can be viewed as an RNN with hidden states S and Z that are updated as follows:

$$S_i = S_{i-1} + \phi(K_i) V_i^\top \tag{15}$$

$$Z_i = Z_{i-1} + \phi(K_i) \tag{16}$$

with initial states $S_0 = Z_0 = 0$.

[10] https://linear-transformers.com/

Hyperparameters	
Layers	1, 2, 3, 4
Gradient clipping	no clip, 0.5, 1
Learning rate	$5e^{-5}, 7e^{-5}, 1e^{-4}$
Batch size	16, 32, 64
Feed-forward dimension	1024, 2048
Self-attention dimension	256, 512

(a) Search space for the reference model

Hyperparameters	
Incremental Processor & Controller	
LSTM layers	1, 2, 3, 4
Controller layers	1, 2
Gradient clipping	no clip, 0.5, 1
Learning rate	$5e^{-5}, 7e^{-5}, 1e^{-4}, 1e^{-3}$
Batch size	16, 32, 64
LSTM hidden dimension	256, 512
Controller hidden dimension	256, 512
Memory size	3, 5, 7
Reviser	
Layers	1, 2, 3, 4
Gradient clipping	no clip, 0.5, 1
Learning rate	$5e^{-5}, 7e^{-5}, 1e^{-4}$
Batch size	16, 32, 64
Feed-forward dimension	1024, 2048
Self-attention dimension	256, 512

(b) Search space for TAPIR

Table 4: Hyperparameter search space for the reference model and TAPIR. The reference model and the reviser share the same search space.

Proof: Duality of the Linear Transformer

Ideally, the information regarding when to revise should be obtained with RNNs, as they have properties that are crucial for incremental processing and therefore can capture high-quality supervision signal. In practice, this is difficult because it cannot perform revision and its recurrence only allows a unidirectional information flow, which prevents a backward connection to any past outputs. For example, creating a link between the input x_t and any past outputs requires computing past hidden states from h_t, which is non-trivial. One technique to achieve this is to use reversible RNNs (MacKay et al., 2018) to reverse the hidden state transition, but this is only possible during training. Another approach involves using neural ODE (Chen et al., 2018) to solve the initial value problem from h_0, which yields h_t for any time step t as the solution, but it would be just an approximation of the true hidden state.

Let us consider an RNN in an incremental scenario, keeping a hidden state h_j. How does x_t affect the *earlier* output y_j for $1 \leq j < t$? We want an answer that satisfies the following conditions for incremental processing:

1. The converse hidden state for time step j computed at time step t, $\ddot{h}_j$, is a function of x_t.

2. The computation of h_t is a function of h_{t-1}, and not of $\ddot{h}_{t-1}$. This is consistent with how RNNs work.

3. The computation of h_{t-1} is valid iff it involves hidden states $h_0, \ldots, h_{t-2}$ that agree with condition (2) in their corresponding step.

In other words, we want a way to compute converse states $\ddot{h}_j$ as a function of x_t, but it should not be affecting h_t, which is only supposed to be computed using past hidden states built from left to right. We are able to satisfy the conditions above and resolve the conflicting hidden state computation by using the Linear Transformer (LT) (Katharopoulos et al., 2020), which can be viewed both as a Transformer and as an RNN. This allows us to get the supervision signal to determine when revision should happen through restart-incremental computation, while still observes how x_t affects all past outputs from the perspective of RNNs.

Let us consider the self-attention computation at time step t for the current and past positions $n, n-1, n-2; n = t$ obtained with a LT under restart-incrementality:

$$\text{Att}^t_n(Q,K,V) = \frac{\phi(Q_n)^\top S_n}{\phi(Q_n)^\top Z_n} \tag{17}$$

$$\text{Att}^t_{n-1}(Q,K,V) = \frac{\phi(Q_{n-1})^\top S_n}{\phi(Q_{n-1})^\top Z_n} \tag{18}$$

$$\text{Att}^t_{n-2}(Q,K,V) = \frac{\phi(Q_{n-2})^\top S_n}{\phi(Q_{n-2})^\top Z_n} \tag{19}$$

From equations (18) and (19) we can see that the hidden state S for computing the representations at positions $n-1$ and $n-2$ are functions of x_n which satisfies condition (1). Furthermore, they are equal to each other *i.e.,* $\ddot{S}_{n-2} = \ddot{S}_{n-1} = S_n = S^t$. Note that we only consider S, however the proof also holds for Z. To satisfy condition (2), consider the self-attention at time step $t-1$ for position $n-1$:

$$\text{Att}^{t-1}_{n-1}(Q,K,V) = \frac{\phi(Q_{n-1})^\top S_{n-1}}{\phi(Q_{n-1})^\top Z_{n-1}} \tag{20}$$

S^t is a function of S^{t-1} in equation (20), $S^t = S^{t-1} + \phi(K_n)V_n^\top$. We also know that $S^t = \ddot{S}_{n-1}$, which means that condition (2) is not completely fulfilled. However, the last clause can be relaxed as it only exists to ensure that the incremental assumption during the computation of S^t is met. The

reason for this is because there are two ways to view the computation of S at any time step t: (1) by updating the previous state S^{t-1}, or (2) computing S^t directly from input tokens $x_1, \ldots, x_{n=t}$, which is analogous to the kernel trick, but in this case S is a combination of projected input tokens. The latter view can be used to relax condition (2), as it means S^t does not completely depend on the previous state (S^{t-1} or $\ddot{S}_{n-1}$) like in conventional RNNs, but can also be computed directly from input tokens while still obeying incremental assumptions.

Fulfilling condition (3) requires that condition (2) holds for all preceding time steps. Formally, $S^i = f(S^{i-1})$ and $S^i \neq f(\ddot{S}); 1 \leq i \leq t-1$. This is satisfied by the fact that $S^i = S^{i-1} + \phi(K_i)V_i^\top$ and taking the perspective of S as a combination of projected input tokens for relaxation. Notice that equation (17) is causal and can be expressed as an RNN at time step t while equations (18) and (19) are acausal. This proof only holds for a *single* layer of LT due to how information flows between layers. Let us consider the computation of S for a multi-layer LT. At time step t, we compute $S^t_{n,l}$ for position $n = t$ in layer l using $x^l_1, \ldots, x^l_n$, which are outputs of layer $l-1$. At the same time, these inputs for layer l are computed using $S^t_{n,l-1}$ from layer $l-1$. This means $x^l_1, \ldots, x^l_{n-1}$ are functions of x^{l-1}_n, which violates the incremental assumption for the input. Therefore, we will be unable to properly examine the effect of the current input on all past outputs if we employ a multi-layer LT.

Algorithm 1 TAPIR

Require: Incremental processor ψ, reviser η, caches $\Gamma^h, \Gamma^z, \Gamma^p$, controller ξ, policy π_θ, input X, input buffer X_{buf}, output buffer Y_{buf}

Initialise: $h_0 \leftarrow 0$, $x_1 \Leftarrow X$, $\tilde{k}_1 \leftarrow 0$, $\tilde{c}_1 \leftarrow 0$, $t \leftarrow 1$
while $t \leq |X|$ **do**
  $h_t \leftarrow \psi(h_{t-1}, x_t)$, $\tilde{y}_t \leftarrow f_\psi(h_t)$, $y_t \leftarrow \text{softmax}(\tilde{y}_t)$
  if $\Gamma^p \neq \emptyset$ **then**
    for $i \leftarrow 1$ to $\min(t-1, N)$ **do**
      $\gamma_i^p \Leftarrow \Gamma^p$, $e_i^t \leftarrow f_\xi(\gamma_i^p, h_t, \tilde{k}_{t-1})$
    end for
    $s^t \leftarrow \text{softmax}(e^t)$, $\tilde{k}_t \leftarrow \sum_i s_i^t \gamma_i^p$, $\tilde{c}_t \leftarrow \sum_i s_i^t c_{i+\max(0,t-N-1)}$
  end if
  $k_t, c_t \leftarrow \xi(\tilde{k}_t, \tilde{c}_t, x_t)$
  $a_t \leftarrow \pi_\theta(k_t)$, $X_{buf} \Leftarrow x_t$
  if $|\Gamma^h| = N$ **then**
    del γ_1^h ▷ Discard the first cache element when full.
  end if
  $\Gamma^h \Leftarrow h_t$ ▷ Update the cache.
  if a_t = `WRITE` **then**
    $Y_{buf} \Leftarrow y_t$, $z \leftarrow f_z(\tilde{y}_t)$, $\varphi \leftarrow f_\phi(h_t, z)$
    if $|\Gamma^z| = N$ **and** $|\Gamma^p| = N$ **then**
      del γ_1^z, **del** γ_1^p
    end if
    $\Gamma^z \Leftarrow z$, $\Gamma^p \Leftarrow \varphi$
  else if a_t = `REVISE` **then**
    $\tilde{y}^\eta_{\leq t} \leftarrow f_\eta(\eta(X_{buf}))$, $Y_{buf} \leftarrow \text{softmax}(\tilde{y}^\eta_{\leq t})$, $\Gamma^z \leftarrow \emptyset$, $\Gamma^p \leftarrow \emptyset$
    for $j \leftarrow \max(1, t-N+1)$ to t **do**
      $h_j \Leftarrow \Gamma^h$, $z \leftarrow f_z(\tilde{y}_j^\eta)$, $\varphi \leftarrow f_\phi(h_j, z)$
      $\Gamma^z \Leftarrow z$, $\Gamma^p \Leftarrow \varphi$
    end for
  end if
  $x_{t+1} \Leftarrow X$, $t \leftarrow t+1$
end while

Tasks/Models	Layers	Clip	Learning Rate	Batch	Feed-forward	Self-attention
Reference model & Transformer reviser						
SF-SNIPS	4	no clip	$1e^{-4}$	16	2048	512
SF-ARW	4	0.5	$1e^{-4}$	16	1024	256
SF-Movie	4	no clip	$5e^{-5}$	16	2048	256
NER-CoNLL	3	1	$1e^{-4}$	64	2048	512
Chunk-CoNLL	3	0.5	$7e^{-5}$	32	2048	512
PoS-CoNLL	3	no clip	$1e^{-4}$	16	2048	512
PoS-UD-EWT	2	-1	$7e^{-5}$	16	2048	512
LT reviser						
SF-SNIPS	3	0.5	$1e^{-4}$	32	1024	512
SF-ARW	4	1	$1e^{-4}$	32	2048	512
SF-Movie	4	0.5	$1e^{-4}$	16	1024	512
NER-CoNLL	3	0.5	$1e^{-4}$	16	2048	512
Chunk-CoNLL	4	0.5	$1e^{-4}$	16	1024	512
PoS-CoNLL	1	no clip	$1e^{-4}$	16	2048	512
PoS-UD-EWT	3	0.5	$7e^{-5}$	16	2048	512

(a) Reference model, Transformer and LT revisers

	Layers					Dimension		
Tasks/Models	LSTM	Ctrl.	Clip	Learning Rate	Batch	LSTM	Ctrl.	Memory
Transformer reviser								
SF-SNIPS	1	1	no clip	$1e^{-3}$	16	512	256	5
SF-ARW	1	2	no clip	$1e^{-4}$	32	512	256	7
SF-Movie	1	1	no clip	$7e^{-5}$	16	512	512	7
NER-CoNLL	3	1	0.5	$1e^{-3}$	16	256	512	3
Chunk-CoNLL	1	2	1	$1e^{-3}$	32	512	256	5
PoS-CoNLL	1	1	no clip	$5e^{-5}$	16	256	256	7
PoS-UD-EWT	1	2	0.5	$1e^{-3}$	16	512	256	3
LT reviser								
SF-SNIPS	2	2	no clip	$1e^{-3}$	64	256	512	3
SF-ARW	1	1	1	$7e^{-5}$	64	256	256	5
SF-Movie	1	1	1	$1e^{-4}$	16	512	256	7
NER-CoNLL	1	1	no clip	$1e^{-3}$	16	256	512	5
Chunk-CoNLL	2	1	1	$1e^{-3}$	16	512	512	7
PoS-CoNLL	1	2	1	$5e^{-5}$	16	256	512	3
PoS-UD-EWT	1	1	no clip	$1e^{-3}$	16	256	256	3

(b) Incremental processor and controller

Table 5: Hyperparameters for our experiments. We use the same hyperparameters for the delayed variants.

Tasks	Dataset	Publication	License	Downloadable
Slot filling	SNIPS	Coucke et al. (2018)	CC0	link / preproc.
Slot filling	Alarm, reminder, & weather	Schuster et al. (2019)	CC BY-SA	link
Slot filling	MIT Movie, eng corpus	Liu et al. (2013)	-	link
NER Chunking PoS tagging	CoNLL-2003	Tjong Kim Sang and De Meulder (2003)	text: NIST research agreement; annotation: -	link
PoS tagging	Universal Dependencies, EWT	Silveira et al. (2014)	CC BY-SA 4.0	link

Table 6: Details about each dataset.

	No. of Seq.					Token Size		Avg. Seq. Length	
Tasks	Train	Valid	Test	Labels	Vocab Size	Train & Valid	Test	Train & Valid	Test
SF-SNIPS	13,084	700	700	72	11,765	124,084	6,354	9.002	9.077
SF-ARW	30,521	4,181	8,621	28	4,215	251,915	62,591	7.259	7.260
SF-Movie	8,797	978	2,443	25	6,710	99,491	24,686	10.178	10.105
NER-CoNLL	14,041	3,249	3,452	9	26,882	254,979	46,394	14.747	13.440
Chunk-CoNLL	14,041	3,249	3,452	23	26,882	254,979	46,394	14.747	13.440
PoS-CoNLL	14,041	3,249	3,452	47	26,882	254,979	46,394	14.747	13.440
PoS-UD-EWT	12,543	2,001	2,077	18	21,917	232,741	25,456	16.003	12.256

Table 7: Descriptive statistics of the datasets. The vocabulary size is computed from training and validation sets.

Tasks	WRITE	REVISE
SF-SNIPS	0.763	0.237
SF-ARW	0.831	0.169
SF-Movie	0.764	0.236
NER-CoNLL	0.904	0.096
Chunk-CoNLL	0.838	0.162
PoS-CoNLL	0.702	0.298
PoS-UD-EWT	0.785	0.215

(a) Training sets

Tasks	WRITE	REVISE
SF-SNIPS	0.794	0.206
SF-ARW	0.838	0.162
SF-Movie	0.772	0.228
NER-CoNLL	0.910	0.090
Chunk-CoNLL	0.790	0.210
PoS-CoNLL	0.819	0.181
PoS-UD-EWT	0.771	0.229

(b) Training and validation sets

Table 8: Mean of WRITE and REVISE action ratios per sentence for training sets and combination of training and validation sets. Most of the time, the mean of the WRITE action ratio is higher compared to the REVISE action ratio.

	Percentage (%) by REVISE Ratio				
Tasks	0-0.2	0.2-0.4	0.4-0.6	0.6-0.8	0.8-1
SF-SNIPS	48.65	30.30	17.56	3.41	0.08
SF-ARW	63.54	23.30	11.58	1.55	0.03
SF-Movie	47.60	34.39	15.63	2.30	0.09
NER-CoNLL	79.67	15.53	4.15	0.64	0.01
Chunk-CoNLL	61.73	27.23	10.21	0.81	0.01
PoS-CoNLL	39.17	24.63	25.33	10.37	0.51
PoS-UD-EWT	50.58	33.24	14.08	2.06	0.05

(a) Training sets

	Percentage (%) by REVISE Ratio				
Tasks	0-0.2	0.2-0.4	0.4-0.6	0.6-0.8	0.8-1
SF-SNIPS	54.80	28.66	13.99	2.52	0.03
SF-ARW	65.01	22.88	10.71	1.35	0.04
SF-Movie	49.21	34.36	14.54	1.85	0.04
NER-CoNLL	81.26	14.75	3.61	0.35	0.03
Chunk-CoNLL	53.20	25.00	18.63	3.13	0.03
PoS-CoNLL	59.35	27.64	11.09	1.84	0.08
PoS-UD-EWT	47.61	32.82	16.81	2.68	0.08

(b) Training and validation sets

Table 9: Distribution of examples in each dataset by their REVISE action ratio for training sets and combination of training and validation sets. Most of the examples in the datasets have considerably low REVISE action ratio (< 0.6).

		TAPIR-Trf			TAPIR-LT		
Tasks	Ref. Model	-	D1	D2	-	D1	D2
SF-SNIPS	91.42	88.26	91.10	90.59	88.12	87.80	88.75
SF-ARW	95.55	94.94	94.96	95.17	93.90	94.63	94.60
SF-Movie	85.20	84.69	84.90	84.30	84.25	84.36	84.33
NER-CoNLL	84.69	80.95	84.52	84.68	82.38	82.90	82.52
Chunk-CoNLL	89.02	88.19	88.86	88.69	85.76	86.92	87.19
PoS-CoNLL	93.07	92.73	92.87	92.81	92.48	92.23	91.98
PoS-UD-EWT	91.88	90.35	91.33	91.67	89.99	90.96	90.69

Table 10: Non-incremental performance of the models on validation sets (F1 for the first group, accuracy for the second group).

		TAPIR-Trf			TAPIR-LT		
Tasks	Ref. Model	-	D1	D2	-	D1	D2
SF-SNIPS	16.2	38.7	38.7	38.7	29.7	29.7	29.7
SF-ARW	4.4	13.6	13.6	13.6	30.7	30.7	30.7
SF-Movie	7.2	21.0	21.0	21.0	25.7	25.7	25.7
NER-CoNLL	16.7	44.7	44.7	44.7	43.7	43.7	43.7
Chunk-CoNLL	16.7	44.1	44.1	44.1	45.2	45.2	45.2
PoS-CoNLL	16.7	42.0	42.0	42.0	33.8	33.8	33.8
PoS-UD-EWT	12.5	34.7	34.7	34.7	38.9	38.9	38.9

Table 11: Number of parameters for each model, in millions.

Tasks/Models	EO	CT	RC	EOΔ1	EOΔ2	RCΔ1	RCΔ2
SF-SNIPS							
Ref. Model	0.181	0.100	0.750	0.081	0.046	0.887	0.933
TAPIR-Trf	0.074	0.055	0.876	-	-	-	-
TAPIR-Trf (D1)	-	0.034	-	0.046	-	0.919	-
TAPIR-Trf (D2)	-	0.030	-	-	0.049	-	0.922
TAPIR-LT	0.081	0.058	0.866	-	-	-	-
TAPIR-LT (D1)	-	0.042	-	0.057	-	0.899	-
TAPIR-LT (D2)	-	0.036	-	-	0.053	-	0.905
SF-ARW							
Ref. Model	0.079	0.041	0.906	0.017	0.008	0.977	0.989
TAPIR-Trf	0.032	0.019	0.950	-	-	-	-
TAPIR-Trf (D1)	-	0.008	-	0.012	-	0.983	-
TAPIR-Trf (D2)	-	0.004	-	-	0.007	-	0.989
TAPIR-LT	0.031	0.019	0.948	-	-	-	-
TAPIR-LT (D1)	-	0.010	-	0.016	-	0.973	-
TAPIR-LT (D2)	-	0.012	-	-	0.019	-	0.972
SF-Movie							
Ref. Model	0.311	0.215	0.549	0.205	0.107	0.705	0.852
TAPIR-Trf	0.079	0.049	0.895	-	-	-	-
TAPIR-Trf (D1)	-	0.028	-	0.043	-	0.930	-
TAPIR-Trf (D2)	-	0.021	-	-	0.033	-	0.944
TAPIR-LT	0.084	0.053	0.889	-	-	-	-
TAPIR-LT (D1)	-	0.041	-	0.069	-	0.906	-
TAPIR-LT (D2)	-	0.033	-	-	0.053	-	0.922
NER-CoNLL							
Ref. Model	0.080	0.057	0.873	0.038	0.023	0.928	0.949
TAPIR-Trf	0.023	0.018	0.931	-	-	-	-
TAPIR-Trf (D1)	-	0.015	-	0.017	-	0.950	-
TAPIR-Trf (D2)	-	0.011	-	-	0.016	-	0.954
TAPIR-LT	0.026	0.019	0.933	-	-	-	-
TAPIR-LT (D1)	-	0.012	-	0.016	-	0.959	-
TAPIR-LT (D2)	-	0.014	-	-	0.017	-	0.947
Chunk-CoNLL							
Ref. Model	0.063	0.026	0.885	0.035	0.027	0.921	0.934
TAPIR-Trf	0.036	0.016	0.905	-	-	-	-
TAPIR-Trf (D1)	-	0.014	-	0.028	-	0.920	-
TAPIR-Trf (D2)	-	0.013	-	-	0.026	-	0.926
TAPIR-LT	0.062	0.026	0.864	-	-	-	-
TAPIR-LT (D1)	-	0.019	-	0.036	-	0.905	-
TAPIR-LT (D2)	-	0.019	-	-	0.041	-	0.901
PoS-CoNLL							
Ref. Model	0.122	0.072	0.788	0.066	0.050	0.859	0.885
TAPIR-Trf	0.042	0.032	0.863	-	-	-	-
TAPIR-Trf (D1)	-	0.025	-	0.033	-	0.890	-
TAPIR-Trf (D2)	-	0.027	-	-	0.037	-	0.884
TAPIR-LT	0.037	0.028	0.871	-	-	-	-
TAPIR-LT (D1)	-	0.023	-	0.031	-	0.898	-
TAPIR-LT (D2)	-	0.026	-	-	0.034	-	0.890
PoS-UD-EWT							
Ref. Model	0.137	0.076	0.796	0.049	0.028	0.907	0.936
TAPIR-Trf	0.053	0.033	0.855	-	-	-	-
TAPIR-Trf (D1)	-	0.023	-	0.033	-	0.907	-
TAPIR-Trf (D2)	-	0.024	-	-	0.034	-	0.905
TAPIR-LT	0.073	0.046	0.818	-	-	-	-
TAPIR-LT (D1)	-	0.030	-	0.041	-	0.888	-
TAPIR-LT (D2)	-	0.024	-	-	0.036	-	0.895

Table 12: Mean of Edit Overhead, Correction Time Score and Relative Correctness. Δt denotes delay for t steps.

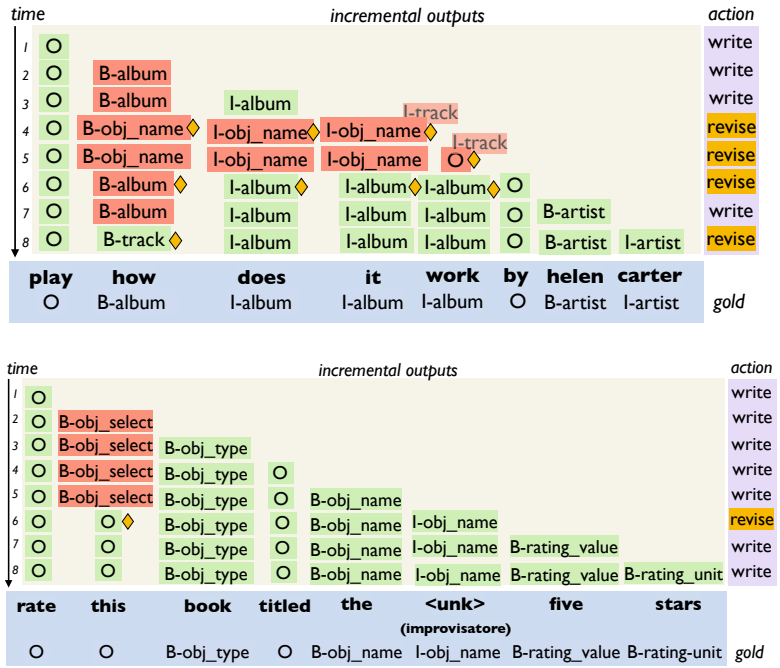

Figure 7: Additional inference examples from SF-SNIPS obtained with TAPIR-Trf. Edited labels are marked by a diamond symbol, with the immediate past output at the top right corner for right-frontier edits. Red labels are incorrect with respect to the final output. In the first example, *how does it* is interpreted as an object name at $t = \{4, 5\}$, but is revised to a part of an album when TAPIR reads *by*. It still makes a mistake at the last step, as it edits the label for *how* from `B-album` to `B-track` when it is unnecessary. TAPIR initially labels *this* in *rate this* as `B-object_select` in the second example, which probably suits the available evidence at $t = 2$. When it encounters the `UNK` token, `B-object_select` is revised to `O`.

Tasks	Overall WRITE	Overall REVISE	Action $\frac{R \cap C}{R}$	Action $\frac{R \cap I}{R}$	Action $\frac{W \cap I}{W}$	Action $\frac{W \cap C}{W}$	Prefix $\frac{R \cap C}{C}$	Prefix $\frac{W \cap C}{C}$	Prefix $\frac{W \cap I}{I}$	Prefix $\frac{R \cap I}{I}$
SF-SNIPS	0.786	0.214	0.606	0.394	0.117	0.883	0.158	0.842	0.521	0.479
SF-ARW	0.802	0.198	0.741	0.259	0.040	0.960	0.160	0.840	0.385	0.615
SF-Movie	0.729	0.271	0.655	0.345	0.075	0.925	0.209	0.791	0.370	0.630
NER-CoNLL	0.896	0.104	0.654	0.346	0.101	0.899	0.078	0.922	0.716	0.284
Chunk-CoNLL	0.773	0.227	0.650	0.350	0.168	0.832	0.187	0.813	0.619	0.381
PoS-CoNLL	0.866	0.134	0.523	0.477	0.234	0.766	0.096	0.904	0.761	0.239
PoS-UD-EWT	0.798	0.202	0.578	0.422	0.257	0.743	0.165	0.835	0.706	0.294

Table 13: Overall distribution of actions and prefixes on test sets using TAPIR-Trf. W represents `WRITE` and R represents `REVISE`. C and I denote correct and incorrect output prefixes, respectively.

	SF-SNIPS				SF-ARW				SF-Movie				NER-CoNLL			
τ	EO	CT	RC	F1	EO	CT	RC	F1	EO	CT	RC	F1	EO	CT	RC	F1
0.0	0.181	0.100	0.750	0.911	0.079	0.041	0.906	0.956	0.311	0.215	0.549	0.840	0.080	0.057	0.873	0.783
0.1	0.102	0.062	0.863	0.905	0.043	0.022	0.944	0.950	0.109	0.066	0.873	0.838	0.040	0.035	0.903	0.761
0.2	0.092	0.059	0.868	0.903	0.039	0.021	0.945	0.946	0.097	0.059	0.881	0.835	0.032	0.026	0.918	0.747
0.3	0.086	0.058	0.870	0.895	0.037	0.020	0.947	0.941	0.090	0.055	0.886	0.832	0.029	0.022	0.925	0.742
0.4	0.079	0.056	0.875	0.888	0.034	0.020	0.948	0.936	0.083	0.051	0.891	0.829	0.025	0.020	0.929	0.742
0.5	0.074	0.055	0.876	0.886	0.032	0.019	0.950	0.934	0.079	0.049	0.895	0.828	0.023	0.018	0.931	0.741
0.6	0.065	0.053	0.878	0.887	0.030	0.018	0.952	0.930	0.074	0.046	0.900	0.826	0.022	0.016	0.935	0.742
0.7	0.055	0.046	0.895	0.875	0.027	0.017	0.953	0.926	0.068	0.043	0.904	0.822	0.021	0.015	0.938	0.743
0.8	0.045	0.041	0.906	0.864	0.024	0.015	0.956	0.918	0.060	0.038	0.914	0.817	0.019	0.014	0.943	0.742
0.9	0.028	0.028	0.934	0.849	0.019	0.013	0.962	0.908	0.049	0.034	0.925	0.804	0.016	0.012	0.949	0.739
1.0	0.000	0.000	1.000	0.814	0.000	0.000	1.000	0.894	0.000	0.000	1.000	0.768	0.000	0.000	1.000	0.700

	Chunk-CoNLL				PoS-CoNLL				PoS-UD-EWT			
τ	EO	CT	RC	F1	EO	CT	RC	Acc	EO	CT	RC	Acc
0.0	0.063	0.026	0.885	0.883	0.122	0.072	0.788	0.923	0.137	0.076	0.796	0.921
0.1	0.054	0.022	0.890	0.878	0.085	0.057	0.816	0.916	0.085	0.049	0.824	0.915
0.2	0.048	0.020	0.895	0.875	0.069	0.047	0.827	0.916	0.075	0.044	0.830	0.913
0.3	0.043	0.018	0.900	0.873	0.059	0.043	0.836	0.915	0.066	0.040	0.840	0.912
0.4	0.039	0.016	0.903	0.871	0.050	0.037	0.850	0.914	0.059	0.036	0.848	0.910
0.5	0.036	0.016	0.905	0.869	0.042	0.032	0.863	0.913	0.053	0.033	0.855	0.908
0.6	0.033	0.015	0.908	0.868	0.035	0.028	0.878	0.912	0.049	0.031	0.861	0.908
0.7	0.031	0.014	0.907	0.867	0.029	0.024	0.893	0.912	0.045	0.029	0.868	0.907
0.8	0.028	0.013	0.911	0.868	0.023	0.020	0.910	0.911	0.040	0.026	0.877	0.906
0.9	0.025	0.012	0.914	0.866	0.016	0.015	0.934	0.909	0.035	0.025	0.885	0.904
1.0	0.000	0.000	1.000	0.833	0.000	0.000	1.000	0.904	0.000	0.000	1.000	0.880

Table 14: Incremental and non-incremental performance of TAPIR-Trf with varying threshold τ for reproducibility purpose.

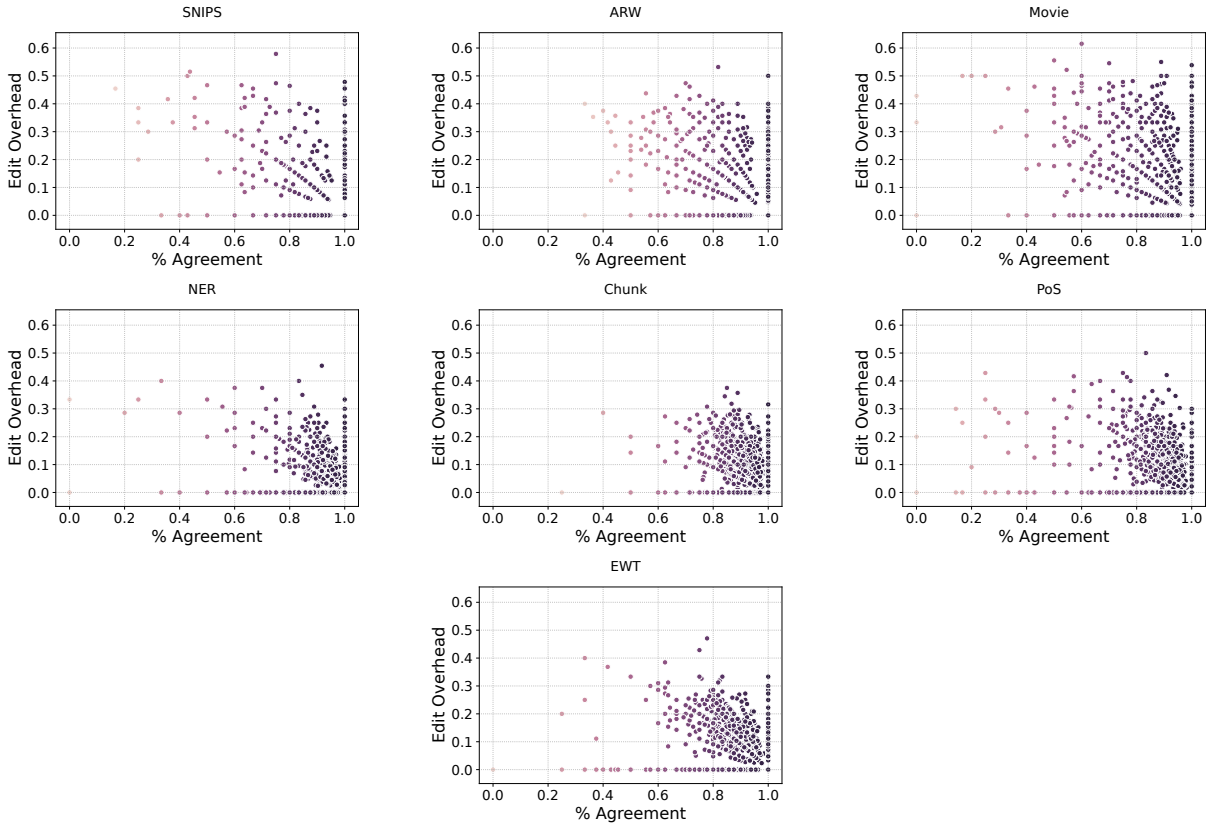

Figure 8: Agreement percentage of the final output between the incremental processor and the reviser of TAPIR-Trf. Disagreements are relatively rare and when they disagree, the range of edit overhead is hardly different compared to the case where both components fully agree with each other.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Section 'Limitations'.

☒ A2. Did you discuss any potential risks of your work?
We don't foresee any particular risks of our work

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Section 'Abstract' and section 1.

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

Section 5

☑ B1. Did you cite the creators of artifacts you used?
Section 5 and Appendix A

☑ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Appendix A

☒ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
We believe its usage in our work is consistent with the intended use for research purposes.

☒ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Please refer to the corresponding publications.

☒ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
We point to the corresponding publications.

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Appendix A

C ☑ Did you run computational experiments?

Section 5 and Appendix A

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Appendix A

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Section 5 and Appendix A

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Section 5

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
In the code repository.

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
Not applicable. Left blank.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
Not applicable. Left blank.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
Not applicable. Left blank.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
Not applicable. Left blank.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
Not applicable. Left blank.