SLATE: A Sequence Labeling Approach for Task Extraction from Free-form Inked Content

Apurva Gandhi[1*], **Ryan Serrao**[2], **Biyi Fang**[1],
Gilbert Antonius[1], **Jenna Hong**[1], **Tra My Nguyen**[3], **Sheng Yi**[4],
Ehi Nosakhare[1], **Irene Shaffer**[1], **Soundararajan Srinivasan**[1], **Vivek Gupta**[5]
Microsoft
{[1]firstname.lastname, [2]ryserrao, [4]shengyi, [3]nguyenm, [5]vivgupt}@microsoft.com

Abstract

We present SLATE, a sequence labeling approach for extracting tasks from free-form content such as digitally handwritten (or "inked") notes on a virtual whiteboard. Our approach allows us to create a single, low-latency model to simultaneously perform sentence segmentation and classification of these sentences into task/non-task sentences. SLATE greatly outperforms a baseline two-model (sentence segmentation followed by classification model) approach, achieving a task F1 score of 84.4%, a sentence segmentation (boundary similarity) score of 88.4% and three times lower latency compared to the baseline. Furthermore, we provide insights into tackling challenges of performing NLP on the inking domain. We release both our code and dataset for this novel task.

1 Introduction

The shift to remote and hybrid working styles due to COVID-19 has led to a large increase in virtual meetings. It has become increasingly important for participants to express themselves and brainstorm as naturally and effortlessly as possible, leading to an opportunity to extract entities from the large amounts of content created in these meetings. A natural entity of interest is a task created by a participant during the meeting which can be assigned to an individual to complete afterwards.

While past works have investigated extracting tasks from typed content such as emails (Bennett and Carbonell, 2005; Wang et al., 2019), there has been less focus on task extraction from more *free-form* content such as digitally handwritten (or *inked*) content on a virtual whiteboard or spoken content in a meeting. Extracting tasks from free-form content is challenging as it is often not as well-structured (e.g., poor grammar, inconsistent/lack of punctuation, typos, etc.) Furthermore, since this content first needs to be converted to text (e.g., through a handwriting recognition model for ink or ASR for speech), downstream NLP models must be robust to errors made by the recognition models.

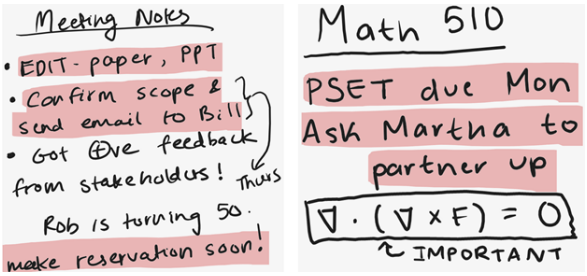


Figure 1: Examples of inked content. Task sentences are highlighted in red. Note the free-form style of the content, containing lists, a paragraph and annotations.

Moreover, as we discuss in Section 2, past approaches for task extraction from typed content assume text to already be segmented into sentences and focus on building a separate sentence-level task classification model. We cannot make this assumption for our scenario since automatic sentence segmentation models trained on typed content are unlikely to generalize well to the inconsistent capitalization and punctuation in free-form content (Stevenson and Gaizauskas, 2000; Rehbein et al., 2020). Furthermore, separating classification and sentence segmentation creates a latency-challenge: Classifying each sentence separately can cause the latency on CPU to scale linearly with the number of sentences, making real-time extraction challenging.

In our work, we address these challenges with our proposed approach SLATE – A Sequence Labeling Approach to Task Extraction from free-form content. Particularly, we apply this approach to inked content, such as that in Fig. 1, and call this application as Ink-SLATE. Our contributions are:

- We create a single, low-latency sequence labeling model to simultaneously perform sentence segmentation and task sentence classification for inked content in a document.
- We leverage ink document layout features to overcome inking domain challenges such as

*Corresponding author.

Proceedings of EMNLP 2022 Industry Track, pages 216–227
December 9–11, 2020. ©2022 Association for Computational Linguistics

the lack of punctuation and capitalization.

- We discuss a custom evaluation procedure suitable for the problem of extracting task sentences from free-form content and benchmark our SLATE approach against a baseline two-model approach.
- We compile a novel dataset for task extraction from ink document text. We release both our code and dataset in our linked repository.[1]

2 Related Work

Past work on analyzing digitally inked content has dominantly fallen into either the category of handwriting recognition (Keysers et al., 2016; Gericke et al., 2012) or document layout analysis – grouping words into document lines (Ye et al., 2005), grouping document lines into blocks of spatially related text (Ye et al., 2007), or detecting indentation (Ye and Viola, 2004). Our work differs from these past works as it analyzes the semantics of the inked content rather than the layout. Furthermore, in an ink-analysis pipeline our work would sit downstream to handwriting recognition and layout analysis rather than replace them; we use, as input to our task extraction system, both recognized text and layout information extracted from the inked document. To our knowledge, our work is the first to tackle both sentence segmentation and task extraction for inked content.

Past work on task extraction has mainly been applied to typed content such as emails (Bennett and Carbonell, 2005; Wang et al., 2019). Other than our different domain of *inked content*, our work also differs in approach from these past works. Particularly, these works assume input text to be already segmented into sentences and focus only on building a classification model to classify these sentences into tasks/non-tasks. Thus, implicitly the task extraction systems in these works rely on a two-model approach: A sentence segmentation model to produce sentence candidates, followed by a classification model to classify sentences as tasks/non-tasks or different sub-categories of tasks. Our approach, on the other hand, uses a single model to simultaneously perform both sentence segmentation and classification for inked content.

Sequence labeling is an NLP approach that predicts a label for each token within a sequence, rather than a label for the whole sequence. Applications of sequence labeling have traditionally included named entity recognition (NER), part-of-speech (POS) tagging, text segmentation, etc. In our work, we reframe the typically two-stage problem of task sentence extraction from documents as a single-stage sequence labeling problem. Since our approach uses a single, shared model to both segment and classify text, it can be thought of as a form of multi-task learning. Multi-task learning has shown to be data-efficient and less prone to overfitting to any single task (Crawshaw, 2020).

Most previous works on sequence labeling use Bi-LSTM and CRF layers in their models (He et al., 2020; Chen et al., 2020). We instead use a RoBERTa architecture, following the recent success of fine-tuning pretrained transformer LMs for data-constrained NLP. (Wolf et al., 2020).

3 Our Approach

3.1 Dataset

Our dataset consists of 200 vendor-created ink documents. To generate these documents, the vendors were provided various example templates with different content (to-do lists, recipes, brainstorms, general notes, etc.) that contain tasks and non-tasks written in various styles (single sentences, paragraphs, lists, diagrams, etc.) For additional diversity, the vendor employed 50+ different donors from different genders and age groups, and with various writing habits (e.g., left/right-handed).

After obtaining these ink documents, we passed them through a handwriting recognition engine (with 9.8% word error rate) and document layout analysis engine similar to ones referred to in Section 2 or to publicly available APIs (Apr, 2022). The result of this is 200 document text blocks and associated layout metadata (line breaks, bullets, etc.; see Section 3.2.4). The layout analysis also groups spatially related regions of text into separate blocks (similar to (Ye et al., 2007)), which we refer to as *writing regions*. We then split these writing regions between 6 annotators who performed two kinds of annotations: (1) Inserting sentence boundaries for sentence segmentation; (2) Annotating each sentence segment as a task/non-task.

Furthermore, the annotators also specially marked certain sentences which were only tasks or non-tasks in the context of the neighboring sentences. An example is a sentence with many misrecognized words, making it incomprehensible in isolation; nevertheless, in the context of a to-do

[1]Dataset, code, and additional details available at: https://github.com/SLATEAuthors/SLATE

list it may become apparent that the sentence is a task. Additional examples of tasks/non-tasks due to context can be found in Appendix A. Table 1 shows dataset split and annotation statistics. For annotation consistency, the annotators were provided a comprehensive annotation guide with example categories of sentences to be labeled as tasks and non-tasks. We release this guide in our linked repository. Since our task is novel, for reproducibility and to support future research, we also make the document texts, layout metadata, and task/sentence annotations available in our repository[1].

Next, we share domain-specific challenges:

Digital handwriting is often misrecognized: We rely on existing handwriting recognition models to convert handwritten strokes to text. Since handwriting is often messy and diverse, recognized text that is inputted to our task extraction model is often ridden with typos and non-sensical words.

Inked content is often overly concise: Users are generally not verbose when inking. Rather, they distil their content to important keywords, phrases, acronyms, phrases, etc. This concise style often lacks proper punctuation and grammar, making NLP tasks such as sentence segmentation to find the boundaries of task/non-task sentences quite challenging. For example, the first inked bullet in Fig. 1 lacks grammar, punctuation and is a list of keywords rather than a proper sentence. Furthermore, the lack of verbosity and the use of esoteric short-hands and acronyms make it even more challenging for a model to understand the meaning of the text. The third bullet in Fig. 1 shows such a shorthand – using '⊕ve' instead of 'positive.'

Ink users want to write in an unrestricted, free-form manner: Inking is conducive to brainstorming. Since people brainstorm tasks in various formats (to-do list, paragraphs, diagrams, mix of these styles, etc.), NLP systems built to analyze inked content must be able to handle this diversity. Fig. 1 shows an example of the free-form nature of inked content. Additional examples are in Appendix A.

3.2 Sequence Labeling Approach

At the heart of SLATE is sequence labeling. A sequence labeling approach treats the input document text as a sequence of tokens (or sub-words). It classifies each token as being part of one of a predefined set of classes. To extract our desired entities (e.g., sentences for sentence segmentation or task sentences for task extraction) we post-process the sequence of tokens according to their predicted class labels. A particular *sequence labeling scheme* determines the set of classes and the logic to post-process the predicted token-level class labels for entity extraction. In this work, we define and try three different sequence labeling schemes, described in the sections below. As will be discussed in Section 3.2.3, sequence labeling is key to letting us simultaneously perform sentence segmentation and task sentence classification with a single model.

Content	Count	
	Train set	Test set
Ink documents	124	83
Sentences	2496	1416
Task sentences	704	440
Non-task sentences	1522	857
Task sentences due to context	173	54
Non-task sentences due to context	97	65

Table 1: Dataset statistics after annotation process.

3.2.1 Sentence-BI Labeling Scheme for Sentence Segmentation

The sentence-BI labeling scheme is used for sentence segmentation and is similar to schemes adopted by past works in sentence segmentation (Rehbein et al., 2020; Le, 2020). In this labeling scheme, tokens are assigned one of two labels: (B) - Beginning of Sentence; (I) - Inside of Sentence.

After the sequence labeling model classifies each token, we aggregate token-level class labels to word-level labels. This is done to make sure that we do not split sentences in the middle of words. The rule used for this aggregation is described in Algorithm 1 of Appendix B. Once we have predicted word-level labels, the words labeled as 'B' indicate the beginning of a new sentence, giving us the predicted sentence boundaries for sentence segmentation as shown in the top row of Fig. 2.

Since a model trained with the sentence-BI labeling scheme is only useful for sentence segmentation, for our task extraction scenario, we need an additional classification model to classify the segmented sentences into tasks/non-tasks. This two-model approach is precisely what we use as our baseline, described in Section 3.3.

3.2.2 SLATE-BIO Labeling Scheme for Task Extraction

The SLATE-BIO labeling scheme is used to extract *task* sentences from the input text. It assigns one

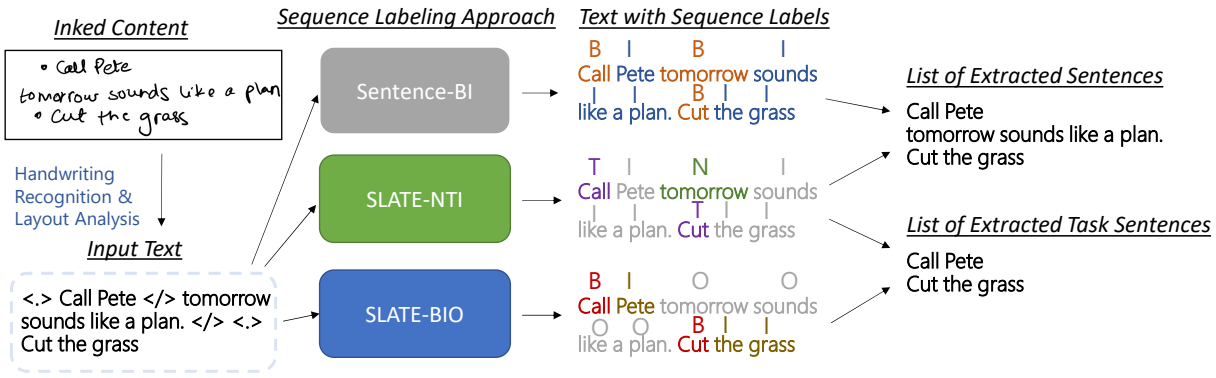


Figure 2: Illustration of the various sequence labeling configurations and how they are used to extract sentences and tasks from inked content.

of the following three labels to each token: (B) - Beginning of Task Sentence; (I) - Inside of Task Sentence; (O) - Outside of Task Sentence. BIO labeling schemes have commonly been used for NER and text chunking tasks (Sang and Buchholz, 2000; Ramshaw and Marcus, 1999). In our work, we adapt it for task sentence extraction.

Similar to sentence-BI, we aggregate predicted token-level labels to word-level labels (described in Algorithm 2 of Appendix B). Once we have the predicted word labels, a sequence of labels that starts with a 'B' and ends in zero or more 'I' labels indicates a task sentence. Task extraction with SLATE-BIO is illustrated at the bottom of Fig. 2.

3.2.3 SLATE-NTI Labeling Scheme for Task and Sentence Extraction

A disadvantage of SLATE-BIO is that while it finds the boundaries surrounding task sentences, it cannot be used for sentence segmentation as it does not find the boundaries between a contiguous block of non-task sentences. For this, we propose the SLATE-NTI labeling scheme to simultaneously train a sequence labeling model for both task sentence extraction and sentence segmentation (a form of multi-task learning). This scheme assigns one of the following three labels to each token: (N) - Beginning of a Non-Task sentence; (T) - Beginning of a Task Sentence; (I) - Inside of a Sentence.

Similar to the other schemes, we aggregate predicted token-level labels to word-level labels (Algorithm 3 of Appendix B). Once we have the predicted word labels, a sequence of word labels that starts with a 'T' and ends in zero or more 'I' labels indicates a task sentence, whereas a sequence that starts with an 'N' and ends in zero or more 'I' labels indicates a non-task sentence. Sentence segmentation and task sentence extraction using SLATE-NTI is illustrated in the middle row of Fig. 2.

3.2.4 Tackling Inking Peculiarities using Document Layout Metadata as Features

As mentioned in Section 3.1, inked content is often written in a casual style, lacking punctuation, proper grammar, verbosity, etc. Furthermore, upstream components like handwriting recognition can introduce errors. This makes modeling especially difficult as standard sentence segmentation relies on punctuation and capitalization to determine sentence boundaries. Similarly, misspelled words, acronyms and improper grammar make it difficult for a model to make sense of the sentence's meaning and thus classify it. To compensate for these peculiarities of inked content, we supplement the input to our sequence labeling model with document layout metadata. Particularly, we add the following to our model input.

Line breaks indicate where a document line ends and a new one begins. While line breaks do not correspond exactly to sentence boundaries, we expect there to be strong correlation between their positions, providing a useful signal to the model for sentence segmentation purposes. We use the '</>' token to indicate a line break in text.

Bullets are used to indicate the start of list items. People tend to write tasks in the form of to-do lists and thus it is common for tasks to be bulleted. Furthermore, bullets almost always indicate the start of a new sentence. Thus, we expect bullets to provide useful signal for both sentence segmentation and task classification. We use the '<.>' token to indicate a bullet in text.

The left side of Fig. 2 shows how we add line breaks and bullets to the model input.

3.3 Baseline Approach

Our baseline approach is a two-model approach where we first train a sentence segmentation model

that takes as input the document text and outputs sentence boundaries. In our work, to build the sentence segmentation model, we use sequence labeling with the Sentence-BI labeling scheme as discussed in Section 3.2.1. We then train a separate sentence classification model that takes as input a sentence and outputs a task/non-task label.

3.4 Model Architecture

For each of the models we train, we fine-tune a pretrained RoBERTa (Liu et al., 2019) encoder implemented in the HuggingFace transformers library (Wolf et al., 2020). For the sequence labeling (SLATE-NTI, SLATE-BIO, and Sentence-BI) approaches, we add classification heads for each input token, to obtain the token-level sequence labels. For the classification model in the baseline approach, we use only a single classification head instead, for classifying one sentence at a time. Leveraging a pretrained transformer model allows us to obtain good performance even with our relatively small training set. For training and implementation details, please refer to Appendix C.

3.5 Evaluation Procedure

A challenge of evaluating SLATE is that since it performs sentence segmentation and classification *jointly*, it is ambiguous how to evaluate its task classification performance. Particularly, since the predicted sentence segments may not match the ground truth sentence segments, it is unclear how to compare their task/non-task labels. For example, consider Fig. 3. At the top, this figure shows sample input text for our task extraction system. The lower left shows the predicted annotation (sentence segmentation and task classification labels) while the lower right shows the ground truth annotation. The predicted task "send email & doc results" has words from two ground truth sentences – "send email & doc" and "results look great." It is unclear which ground truth sentence we should compare its classification label with. Thus, without an explicit matching from predicted task segments to ground truth segments, it is ambiguous how to compare the labels of predicted and ground truth sentences. In our work, we use a bipartite graph matching algorithm to construct such an explicit matching using IOU similarity as edge weights between predicted and ground-truth sentences (Section 3.5.1). The result of this procedure is an explicit matching, allowing us to port typical classification metrics to our scenario. For example, Fig. 3 shows how we

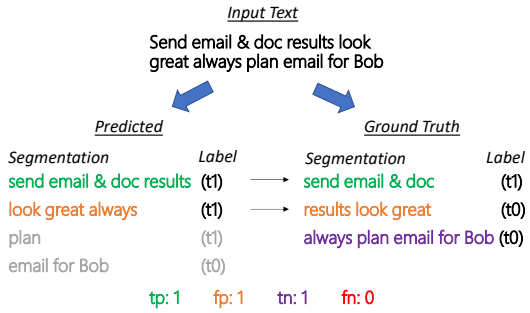


Figure 3: Example of calculating the number of true positives (tp), false positives (fp), true negatives (tn) and false negatives (fn) for the given input text, predictions and ground truth annotations. The t0/t1 labels are used as abbreviations for non-task/task labels.

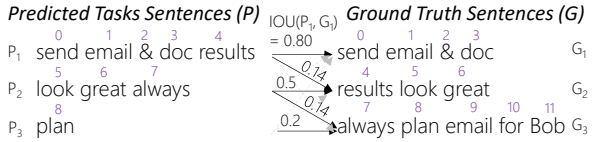
can calculate true/false positives/negatives, from which further classification metrics (e.g., accuracy, F1, etc.) can be computed. Additional discussion on our evaluation procedure and comparison to standard NER metrics is provided in Appendix D.

For evaluating the sentence segmentation performance of SLATE, we use the *boundary similarity (B)* sentence segmentation metric introduced in (Fournier, 2013). Concisely, *B* penalizes a predicted segmentation based on the number of edits required to transform the predicted segmentation to the ground truth segmentation. Near boundary misses are penalized less compared to full misses/additions. *B* is a score from 0-1 where a higher score represents a better predicted segmentation and a 1 represents a perfect segmentation. More metric details are in Appendix D.3.

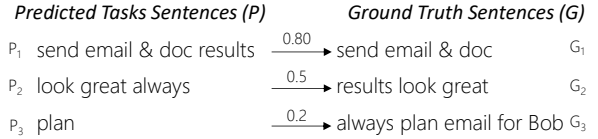
In our application, since the segmentation quality of extracted task sentences matters more than that of non-task sentences, we also compute a modified version of *B* which we call the *true positive boundary similarity (B_{tp})*. The formula to compute B_{tp} is the same as Equation 2 (Appendix D.3) except that in the segmentations that we compare, we only include the boundaries of true positive tasks.

3.5.1 Matching Predicted Task Sentences to Ground Truth Sentences

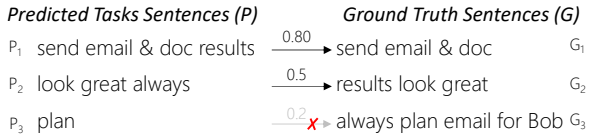
In this section, we describe the procedure used to obtain a matching between predicted task sentences and ground truth sentences. Let D represent the document text provided to the model to perform inference on. D is then a sequence of words w_i where $index(w_i)$ represents the positional index of w_i in D. Let G be a partition of D corresponding to the ground truth sentences (task and non-task) in D, i.e, each element $G_i \in G$ is a set of words



(a) Step 1 of the matching process constructs a complete weighted bipartite graph between the sets of predicted task sentences and ground truth sentences. The edge weights represent the IOU similarity between sentences, calculated on their respective sets of word indices (purple). The light gray edges have zero edge weights.



(b) Step 2 of the matching process finds the maximum weight full matching for the constructed graph.



(c) Step 3 of the matching process prunes out edges that do not meet a minimum similarity threshold.

Figure 4: Illustration of the procedure used to match predicted task sentences to ground truth sentences.

$w_j \in D$ representing a sentence in the ground truth segmentation of D. After performing inference with our model on D we observe the set P corresponding to the set of predicted task sentences, i.e, each element $P_i \in P$ is a set of words $w_j \in D$ representing a predicted task sentence. With this notation, we are now ready to describe the steps of the matching procedure:

1. **Construct a complete weighted bipartite graph between the sets P and G where each element (sentence) in the sets is a node and edge weights represent similarity between the node sentences.** A complete bipartite graph between sets P and G is one where every pair of nodes from differing sets have an edge but no pair of nodes from the same set has an edge. We use the Intersection over Union (IOU) between sentences in the graph to measure similarity. In our scenario, we define IOU as follows:[2]

$$IOU(P_i, G_j) = \frac{|P_{i-ind} \cap G_{j-ind}|}{|P_i \cup G_j|}$$

$$\text{where } S_{-ind} \coloneqq \{index(w_l) \mid w_l \in S\}. \quad (1)$$

2. **Find the maximum weight full matching M for the bipartite graph constructed above.** We desire a matching between the sets P and G to maximize the overall similarity between matched sentences. Let $C = ((P, G), E)$ represent the weighted complete bipartite graph we constructed in first step, where the set E represents the set of edges in the graph and $weight(e)$ for $e \in E$ represents the IOU similarity score between the nodes that e connects. We construct a matching $M \subseteq E$ such that each node in the graph is included in at most one edge in M and so that $|M| = min(|P|, |G|)$. When $|P| = |G|$, this is commonly known as a perfect matching. Since in our scenario we allow $|P|$ and $|G|$ to differ, we follow Karp (1980) and refer to this as a *full* matching. Furthermore, we choose the edges in M so as to maximize $\sum_{e \in M} weight(e)$, making M the maximum weight full matching for graph C. Essentially, M chooses the edges between predicted task sentences and ground truth sentences that maximizes the overall similarity between matched sentences. Also, note that the full matching allows at most one predicted segment to be matched to a ground truth sentence, preventing overcounting in the case where there are more than one predicted sentences for a single ground truth sentence.

3. **Prune M to remove matches that have non-significant overlap.** To provide credit only when predicted tasks are sufficiently similar to ground truth task sentences, we define a threshold t and remove edges $e \in M$ with $weight(e) < t$. In our work, we set $t = 0.25$.

Fig. 4 illustrates the matching procedure steps.

4 Results

The performance of the various modeling approach configurations are presented in Table 2. The first four rows show the performance of the different SLATE configurations tried and the last row shows the performance of the baseline approach. Our flagship approach is SLATE-NTI which has the following advantages: (1) The highest segmentation performance (B and B_{tp}); (2) Much lower latency compared to the baseline approach; (3) Better or at least comparable task classification performance with respect to the other approaches. The remainder of this section discusses some observed trends.

[2] Using $P_{i-ind} \cap G_{j-ind}$ in the numerator of the definition of $IOU(P_i, G_j)$ instead of simply $P_i \cap G_j$ is important to avoid spurious matches when the same words may be repeated more than once in D.

Method	Task (%)				Non-task (%)				Acc (%)	B_{tp} (%)	B (%)	Latency (ms)
	Rec	Prec	F1	Context Rec	Rec	Prec	F1	Context Rec				
SLATE-NTI with Doc Metadata	87.4	81.7	84.4	69.6	89.3	92.9	91.1	60.9	88.7	89.1	88.4	34.2
SLATE-BIO with Doc Metadata	88.9	80.4	84.4	75.6	88.3	93.6	90.8	64.0	88.5	86.6	-	
SLATE-NTI	90.2	81.2	85.5	78.1	88.6	94.4	91.4	59.1	89.2	84.3	83.4	26.5
SLATE-BIO	87.4	83.2	85.3	70.4	90.4	93.0	91.7	63.7	89.4	83.0	-	
Baseline	83.1	81.5	82.3	43.0	89.8	90.1	90.2	57.8	87.4	85.5	85.3	90.6

Table 2: Performance comparison of the various modeling approach configurations on our test set. The classification (recalls, precisions, F1s and accuracy) and segmentation metrics (B_{tp} and B) are averages over five distinctly seeded training runs for the corresponding modeling method. The latency values were obtained by first finding the mean latency of the model inference over each sample in the test set and then performing the average of five such runs.

4.1 The Latency Advantage of SLATE

As shown in Table 2, the SLATE approaches have 2.6 to 3.4 times lower inference latency compared to the the baseline approach, depending on whether they use document metadata or not. This lower latency of SLATE can be attributed to two main reasons. The first is that SLATE uses a single model for both segmentation and classification, whereas the baseline approach suffers from the combined latency of two separate models. The second reason is that since the classification model of the baseline approach acts on each sentence independently, its inference time scales linearly with the number of sentences in the input. The SLATE approach on the other hand can perform inference on an input containing multiple sentences in a single inference.

4.2 SLATE Benefits from Context

Unlike the sentence classification model in our baseline which has access to only a single sentence per inference, the SLATE approach has access to contextual information since it performs inference on a whole block of text at once. Table 2 shows the advantage that access to context gives SLATE compared to the baseline approach: Every SLATE approach has better classification performance (F1 scores and Accuracy) compared to the baseline. To further zoom in on the effect of contextual information, Table 2 also shows the recall of the approaches on sentences in the test set that were annotated as being tasks or non-tasks only due to context (see the Context Rec. columns). We see that each of the SLATE approaches has significantly higher recalls compared to the baseline on such sentences.

4.3 The Benefit of Multi-task Learning

SLATE with either the BIO or NTI labeling schemes is inherently a form of multi-task learning as it is trained to both segment text and classify segmented text simultaneously. Still, SLATE-NTI has a stronger multi-task component compared to SLATE-BIO since unlike the BIO scheme, the NTI scheme forces the model to not only learn how to segment out task sentences but non-task sentences as well. Learning how to find the boundaries of non-task sentences is complementary to learning to find boundaries of task sentences. Thus, SLATE-NTI learns to be more effective at segmenting the text compared to SLATE-BIO. This can be seen in Table 2 by observing that the B_{tp} scores for SLATE-NTI configurations are higher than their corresponding SLATE-BIO configurations.

4.4 Adapting to Ink using Layout Metadata

As discussed in Section 3.2.4, we expect that adding document layout information such as line breaks and bullets to the model input should help compensate for the lack of traditional characteristics of natural language such as proper grammar, punctuation, capitalization, verbosity, etc. The results in Table 2 substantiate this expectation as we see large margins of improvement (> 3.6%) in the segmentation metrics (B and B_{tp}) when we compare the SLATE approaches that use document metadata against those that do not. Thus, supplementing the model with document layout information is an effective method to adapt to the segmentation challenges of the inking domain.

5 Conclusion

We have presented SLATE, a single-model, sequence labeling approach for extracting tasks from free-form content. It overcomes ink domain challenges via our custom ink dataset and ink-document layout information. Our flagship configuration, SLATE-NTI, is a single, low-latency model trained for both accurate sentence segmentation and task sentence classification on inked content.

References

2022. Cross-platform handwriting recognition and interactive ink apis. https://developer.myscript.com/.

Paul N Bennett and Jaime Carbonell. 2005. Detecting action-items in e-mail. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 585–586.

Luoxin Chen, Weitong Ruan, Xinyue Liu, and Jianhua Lu. 2020. Seqvat: Virtual adversarial training for semi-supervised sequence labeling. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8801–8811.

Nancy Chinchor and Beth Sundheim. 1993. MUC-5 evaluation metrics. In *Fifth Message Understanding Conference (MUC-5): Proceedings of a Conference Held in Baltimore, Maryland, August 25-27, 1993*.

Michael Crawshaw. 2020. Multi-task learning with deep neural networks: A survey. *arXiv preprint arXiv:2009.09796*.

Chris Fournier. 2013. Evaluating text segmentation using boundary edit distance. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1702–1712.

Chris Fournier and Diana Inkpen. 2012. Segmentation similarity and agreement. *arXiv preprint arXiv:1204.2847*.

Lutz Gericke, Matthias Wenzel, Raja Gumienny, Christian Willems, and Christoph Meinel. 2012. Handwriting recognition for a digital whiteboard collaboration platform. In *2012 International Conference on Collaboration Technologies and Systems (CTS)*, pages 226–233. IEEE.

Zhiyong He, Zanbo Wang, Wei Wei, Shanshan Feng, Xianling Mao, and Sheng Jiang. 2020. A survey on recent advances in sequence labeling from deep learning models. *arXiv preprint arXiv:2011.06727*.

Richard M Karp. 1980. An algorithm to solve the m× n assignment problem in expected time o (mn log n). *Networks*, 10(2):143–152.

Daniel Keysers, Thomas Deselaers, Henry A. Rowley, Li-Lun Wang, and Victor Carbune. 2016. Multi-language online handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

The Anh Le. 2020. Sequence labeling approach to the task of sentence boundary detection. In *Proceedings of the 4th International Conference on Machine Learning and Soft Computing*, pages 144–148.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Chunping Ma, Huafei Zheng, Pengjun Xie, Chen Li, Linlin Li, and Luo Si. 2018. Dm_nlp at semeval-2018 task 8: neural sequence labeling with linguistic features. In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 707–711.

Lance A Ramshaw and Mitchell P Marcus. 1999. Text chunking using transformation-based learning. In *Natural language processing using very large corpora*, pages 157–176. Springer.

Ines Rehbein, Josef Ruppenhofer, and Thomas Schmidt. 2020. Improving sentence boundary detection for spoken language transcripts. In *Proceedings of the 12th International Conference on Language Resources and Evaluation (LREC), May 11-16, 2020, Palais du Pharo, Marseille, France*, pages 7102–7111. European Language Resources Association.

Erik F Sang and Sabine Buchholz. 2000. Introduction to the conll-2000 shared task: Chunking. *arXiv preprint cs/0009008*.

Erik F Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050*.

Isabel Segura Bedmar, Paloma Martínez, and María Herrero Zazo. 2013. Semeval-2013 task 9: Extraction of drug-drug interactions from biomedical texts (ddiextraction 2013). Association for Computational Linguistics.

Mark Stevenson and Robert Gaizauskas. 2000. Experiments on sentence boundary detection. In *Sixth Applied Natural Language Processing Conference*, pages 84–89.

Julien Tourille, Matthieu Doutreligne, Olivier Ferret, Aurélie Névéol, Nicolas Paris, and Xavier Tannier. 2018. Evaluation of a sequence tagging tool for biomedical texts. In *proceedings of the Ninth International Workshop on Health Text Mining and Information Analysis*, pages 193–203.

Wei Wang, Saghar Hosseini, Ahmed Hassan Awadallah, Paul N Bennett, and Chris Quirk. 2019. Context-aware intent identification in email conversations. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 585–594.

Thomas Wolf, Julien Chaumond, Lysandre Debut, Victor Sanh, Clement Delangue, Anthony Moi, Pierric Cistac, Morgan Funtowicz, Joe Davison, Sam Shleifer, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45.

Ming Ye, Herry Sutanto, Sashi Raghupathy, Chengyang Li, and Michael Shilman. 2005. Grouping text lines in freeform handwritten notes. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 367–371. IEEE.

Ming Ye and Paul Viola. 2004. Learning to parse hierarchical lists and outlines using conditional random fields. In *Ninth International Workshop on Frontiers in Handwriting Recognition*, pages 154–159. IEEE.

Ming Ye, Paul Viola, Sashi Raghupathy, Herry Sutanto, and Chengyang Li. 2007. Learning to group text lines and regions in freeform handwritten notes. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 1, pages 28–32. IEEE.

A Ink Document Examples

Here we provide additional examples of task/non-task sentences occurring in various styles in ink documents. Fig. 5 is an example of a to-do list style ink document. Here we see task sentences mainly written in the form of bullets some of which also span over multiple lines. Note certain sentences are tasks only based on the context and might not seem like task sentences otherwise.

Similarly, Fig. 6 shows an example of an inked recipe content. Although some of the sentences may seem like tasks, they are not in the context of being a recipe. For example, while "add tomato and garlic to make sauce" may be written like a task sentence, it is not considered a task sentence as it is an instruction in a recipe.

B Token-level to Word-level label Aggregation Rules for Sequence Labeling

Algorithm 1: Sentence-BI rule for aggregating token-level labels to word-level labels.

Input : $tokenLabels$ is a list of a token-level labels for a given word.
Output : Word-level label aggregated from $tokenLabels$.

```
if 'B' in tokenLabels then
    return 'B';
else
    return 'I';
end
```

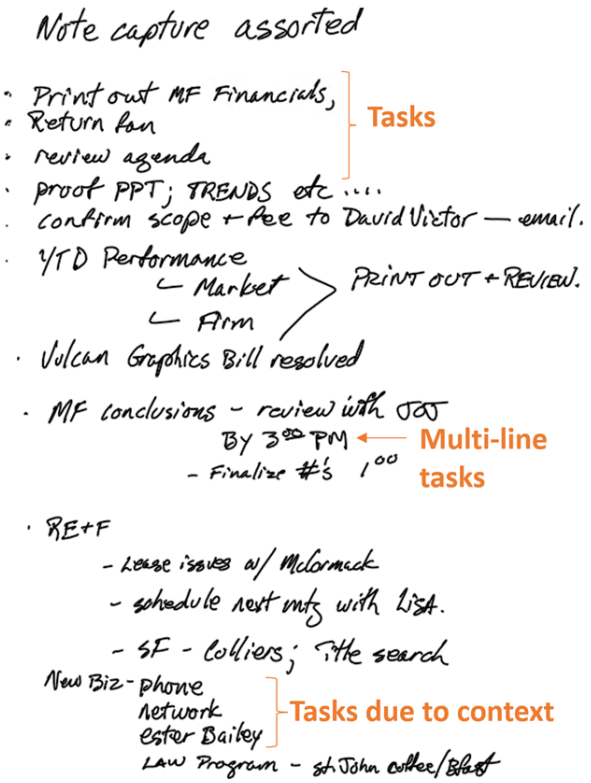


Figure 5: Example of task sentences in an ink document.

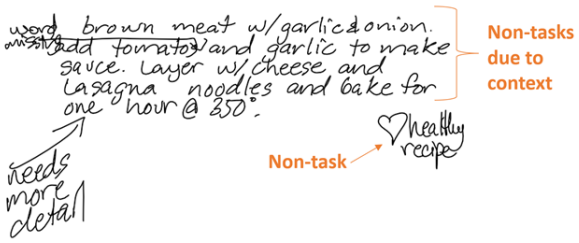


Figure 6: Example of non-task sentences in an ink document.

Algorithm 2: SLATE-BIO rule for aggregating token-level labels to word-level labels.

Input : $tokenLabels$ is a list of a token-level labels for a given word.
Output : Word-level label aggregated from $tokenLabels$.

```
if 'B' in tokenLabels then
    return 'B';
else
    return mode(tokenLabels);
end
```

Algorithm 3: SLATE-NTI rule for aggregating token-level labels to word-level labels.

Input : $tokenLabels$ is a list of a token-level labels for a given word.
Output : Word-level label aggregated from $tokenLabels$.

```
if 'N' or 'T' in tokenLabels then
    if # of 'T' labels > # of 'N' labels then
        return 'T';
    else
        return 'N';
    end
else
    return 'I';
end
```

C Training, Implementation and Hyperparameter Details

Each of our models were implemented using the HuggingFace Transformers library (Wolf et al., 2020). For sequence labeling models, we use the $AutoModelForTokenClassification$ class with the $RoBERTa_{base}$ architecture. For our sentence classification model in the baseline, we use the $AutoModelForSequenceClassification$ class with the $RoBERTa_{base}$ architecture. The model encoders were initialized using the pretrained weights provided by the library.

For training we use a batch size of 3 and 16 for the sequence labeling models and classification model respectively. All models were fine-tuned on our training set for 100 epochs. The objective for all the models was a class-weighted cross-entropy loss. The learning rate was kept constant at 1×10^{-6} for all of the models.

While training we use a machine with an NVIDIA RTX 2080ti GPU, Intel i9-9900K CPU and 64GB of RAM. For latency experiments, inference was done on the CPU of the above machine.

D Evaluation Procedure

D.1 Our Evaluation Approach

Our evaluation procedure has the following steps:

1. Match predicted task sentences to ground truth sentences (task and non-task) that have significant overlap. The matching procedure is described in more detail in Section 3.5.1.
2. Calculate the number of true/false positives/negatives according to the following definitions:

   True Positive: Predicted task sentence that is matched to a ground truth task sentence.

   False Positive: Predicted task sentence that is matched to a ground truth non-task sentence.

   True Negative: Ground truth non-task sentence that is not matched to any predicted task sentence.

   False Negative: Ground truth task sentence not matched to any predicted task sentence.

   Fig. 3 shows the calculation of true/false positives/negatives for a sample input text, prediction and ground truth annotation.
3. Calculate standard classification metrics (accuracy, recall, precision, f1-scores, etc.) from the true/false positives/negatives to understand the task classification performance. When calculating non-task recall/precision, we treat true/false positives as true/false negatives and vice versa.
4. Calculate sentence segmentation metrics to evaluate the quality of the extracted task sentences. In this work we use the *boundary similarity (B)* sentence segmentation metric introduced in (Fournier, 2013). The boundary similarity metric is based on the *boundary edit distance (BED)* introduced in (Fournier and Inkpen, 2012). Concisely, the boundary similarity metric penalizes a predicted segmentation based on the number of edits required to transform the predicted segmentation to the ground truth segmentation. Near boundary misses are penalized less compared to full boundary misses/additions. *B* is a score from 0-1 where a higher score represents a better predicted segmentation and a 1 represents a perfect match to the ground truth segmentation. More metric details are in Appendix D.3.

   In our application, since the segmentation quality of extracted task sentences matters more than that of non-task sentences, we also compute a modified version of this metric which we call the *true positive boundary similarity (B_{tp})*. The formula to compute B_{tp} is the same as Equation 2 (Appendix D.3) except that in the segmentations that we compare, we only include the boundaries of true positive tasks in the predicted segmentation and

the corresponding boundaries for the matched tasks in the ground truth segmentation.

D.2 Why not use standard NER Evaluation instead?

An alternative approach could be to leverage Named Entity Recognition (NER) metrics where the entities we are trying to recognize are task sentences. But these metrics are not without issues either. NER systems are typically evaluated by calculating precision, recall, and F1-scores at either the token level (Ma et al., 2018; Tourille et al., 2018) or at the entity level (Sang and De Meulder, 2003; Segura Bedmar et al., 2013). Token-level metrics suffer from being difficult to interpret compared to entity-level metrics. However, entity-level metrics are often too strict, giving credit only when predicted entities match the ground truth exactly (Sang and De Meulder, 2003). For example, in our scenario, if our model misses only a single token in an extracted task sentence, it would get no credit.

To address this, other evaluation schemes that provide credit for partial entity matches have been proposed (Segura Bedmar et al., 2013; Chinchor and Sundheim, 1993), but these tend to be more complex and difficult to interpret compared to the standard sentence classification metrics. In this work, we propose an evaluation procedure that allows us to calculate metrics that can be interpreted in the same way that standard sentence classification and segmentation metrics are, but at the same time provides enough slack to allow partial matches of predicted and ground truth task sentences.

D.3 The Boundary Similarity (B) Metric

Let us define a segmentation S of text to be a sequence of boundary positions where each boundary represents where a sentence begins and/or ends. A boundary can be placed between words and by default we place boundaries before the first word and after the last word in the text. Boundary edit distance (BED) is a measure of the minimum number of edits that need to be made to a given segmentation S_1 to make it identical to another segmentation S_2. There are three types of edit operations we can make to S_1 in order to bring it into parity with S_2:

- **Addition (A):** When S_1 is missing a boundary that is in S_2 we can add a boundary to S_1.
- **Deletion (D):** When S_1 has a boundary where S_2 does not, we can delete this boundary from S_1.
- **n-wise Transposition (T):** When S_1 misses a boundary in S_2 but has one in the near neighborhood, instead of making two edits to S_1 (one A and one D operation), we allow a single T operation which involves transposing/shifting the near boundary in S_1 to the corresponding position in $S2$. The parameter n determines how far boundaries in S_1 and S_2 can be to be considered for a T operation instead of an A or D operation. In this work, we set $n = 2$, allowing transpositions when boundaries differ by a maximum of 2 positions.

Suppose we calculate the BED for two segmentations of the same text S_1 and S_2. The BED outputs the following: N_M is the number of perfect matches between boundaries, requiring no edits; N_A is the number of A operations required; N_D is the number of D operations required; the set $S_t = \{t \mid t = \#\text{ positions a boundary should be shifted } \forall\ T \text{ operations required}\}$. Then *boundary similarity (B)* between S_1 and S_2 is calculated as follows:

$$B(S_1, S_2) = 1 - \frac{N_A + N_D + \sum_{t \in S_t} \frac{t}{n}}{N_M + N_A + N_D + |S_t|} \quad (2)$$

Essentially, B gives no credit when a boundary is completely missed (A or D operation required) but gives partial credit when a near miss occurs (T operation required). For a more detailed explanation of this metric you may refer to (Fournier, 2013).

E Limitations

Here we discuss limitations of the work. First, since there are no past works and open datasets in the literature for our task, we are unable to benchmark against past works directly. To help address this, we have decided to open-source our dataset, modeling code, and evaluation code, so future research works can leverage these for benchmarking purposes. Still, this dataset is not very large. It consists of roughly 200 annotated ink documents. While this gave us a decent number of task and non-task sentences for fine-tuning a pretrained model and evaluating it on our task, with more data there are other approaches that we could take to further supplement our approach. For example, while we do try to address domain-specific noise such as errors introduced by handwriting recognition or poor grammar in inked content using document layout information, with a larger ink document corpus, we

could try supplementing our methodology for domain adaptation with language modeling. Finally, while we work with free-form content like inked documents, our work assumes the input to be recognized text from this content rather than the raw content (ink strokes). For example, handwriting recognition and document layout analysis methods are out of scope for this work. We cite examples of other works in literature and APIs that deal with these components in Sections 2 and 3.1.