Master thesis
Computing Science

Radboud University University of Glasgow

Extracting Structured Web Content using Deep Neural Language Models

Finding order amidst the chaos

Author:
Gijs Hendriksen
s4324544

First supervisor/assessor:
Prof.dr.ir Arjen de Vries
a.devries@cs.ru.nl

Second supervisor:
Dr Jeff Dalton
jeff.dalton@glasgow.ac.uk

Second assessor:
Dr.ir Faegheh Hasibi
f.hasibi@cs.ru.nl

August 23, 2022

Acknowledgements

I would like to express my deepest appreciation to both of my supervisors, Jeff Dalton and Arjen de Vries, for their invaluable guidance and feedback. I am also extremely grateful to the other members of GRILL Lab at the University of Glasgow - Sophie Fischer, Carlos Gemmell, Iain Mackie, Paul Owoicho, Federico Rossetto, Ivan Sekulić and Alessandro Speggiorin - for sharing their expertise, providing me with new insights, assisting me wherever possible and generally making my time in Glasgow an amazing experience. Finally, I would like to thank my girlfriend, Kimberley Frings, for proofreading and providing feedback on parts of my thesis and motivating and supporting me throughout the entirety of the project.

Abstract

Structured information extraction from the Web plays an important role in many large-scale automated systems, required to enable varying downstream applications. In this thesis, we propose an information extraction method leveraging deep neural language models to extract entity attributes from HTML documents. Our proposed model traverses the DOM tree to split the page into subtrees and encodes these into sequences capturing the HTML structure. It then performs attribute extraction on each of these sequences, reranking the results to obtain a final prediction. We evaluate our models on the SWDE dataset and obtain results on par with the state of the art. We argue that the segmentation of a page into several smaller sequences helps the model distil structured information from a web page, and empirically show that the inclusion of the HTML structure in the sequence increases performance.

Contents

Chapter 1

Introduction

The internet has become a massive source of information. Be it recipes, medical information or random trivia, almost anything you can think of can nowadays be found on the Web. With all this knowledge readily available, automated systems that require vast amounts of data - such as personal assistants or conversational agents - can be improved like never before.

However, most of this information is contained in *semi-structured content*. Although the layout of a web page is structured using HTML, the actual content of the page is written by people for other people to read, and as such mostly consists of natural language. How much information an automated system can use from any given website depends on its ability to distil the relevant information on a page into a *structured* format it can use.

To make this task easier, many websites are now incorporating structured data markups in their websites. They use a vocabulary defining classes and relations for many different types of entities from different domains, and annotate their HTML markup in such a way that an automated system can easily detect which entities occur on the page and what their properties are. To standardise usage of structured data markups across the Web, the Schema.org vocabulary [11] was introduced, which has since been adopted by many different websites.

As of October 2021, roughly 40% of websites are using structured data markups on their web pages [1]. These structured pieces of data power many downstream applications, the most well-known being info boxes in search results or email confirmations. However, more than half of the websites on the internet do not supply these structured data markups yet. And, because it takes some level of technical knowledge to implement these formats in a website, it is unlikely we will ever reach a point where all information on the Web will be made available in such a structured fashion.

In this thesis, we propose another method of extracting structured content from semi-structured web pages. Our method does not rely on annotations created and maintained by the owner of the website. Instead, we make use of recent advances in pre-trained neural language models such as BERT [9] to understand segments of a page and detect relevant pieces of data automatically. We evaluate our model on the Structured Web Data Extraction (SWDE) dataset [13] and compare our results against several baseline architectures. We also investigate how well our model can adapt to new, unseen websites once it has been trained on a separate set of websites.

Contrary to existing work, our models use a simple algorithm to split a webpage into segments and represent each of these segments as a sequence of text. We investigate the following research questions in depth (Chapter 3):

- How can we leverage powerful neural language models to enable effective attribute extraction?
- How does the size of the HTML segment influence the performance of the model?
- How can we encode the structure of the HTML segment into sequences of text in a meaningful and effective way?
- How well do our models generalise to new, unseen websites?

With our setup, we are able to fine-tune off-the-shelf neural language models without changing the architecture of the model or adjusting the input embeddings (Chapter 4). Though this approach works well on the standard information extraction task, our models have trouble adjusting to unseen websites in the zero-shot setting (Chapter 5).

Chapter 2

Background

2.1 Neural language models

Over the last decade, deep learning has been used extensively for a wide variety of tasks. Neural networks have shown to perform very well on tasks with numerical input data. For instance, convolutional neural networks have significantly advanced the area of computer vision, and recurrent neural networks have been used widely to handle data of sequential nature, such as time series. However, it is not as straightforward to efficiently translate natural language processing tasks to a deep learning approach.

2.1.1 Representation

The first challenge of posing natural language processing as a deep learning task is representation. Since machine learning models work on numeric data, we somehow need to transform the input text. A basic approach treats the input sequence as a 'bag-of-words', by simply counting occurrences of each token in the sequence and representing those counts as a vector. However, the dimensionality of these vectors quickly becomes very large and the vectors themselves sparse, as each vector has to contain counts for each token in the vocabulary, and each sequence contains only a small number of distinct tokens. More importantly, this representation disregards all structure and grammar the original text contained, which severely limits its expressiveness.

Another representation, which would keep the order of the text intact, is to 'one-hot' encode each token in the input sequence and stack these encodings to obtain the full representation of the sequence. Again, each vector contains a position for each token in the vocabulary, where the value of that position is 1 if the encoded token is equal to that vocabulary token, and 0 otherwise. Thus, for a vocabulary of size V and a sequence of length N, the one-hot

representation of the input sequence would be a $N \times V$ matrix with a single 1 on each row, and a 0 in all other positions.

One-hot encoded text also suffers from large and sparse input vectors (even more so than the bag-of-words approach), as each token vector must be as long as the size of the vocabulary. This could in theory be solved by limiting the size of the vocabulary, but that would mean we can no longer represent all possible input sequences. To limit the size of the input embeddings, but still be able to represent all inputs, we can tokenise the input into *subword units*. With this approach, less common words are split into subcomponents. For instance, the word 'amazingly' can be split into tokens `["amazing", "##ly"]`, where the `##` characters indicate that the token is connected to the previous token. To generate the vocabulary, one usually starts with a vocabulary of single characters, and then repeatedly finds likely token combinations (A, B) and replaces both tokens with a single token AB. This procedure is repeated until the desired vocabulary size is reached.

The subword unit approach is used in most modern neural language models. Examples of tokenisation methods that use this technique are Byte Pair Encoding (BPE) [31, 30], WordPiece [36] and SentencePiece [18].

Besides ensuring all words can be represented using a limited vocabulary size, the subword approach has an additional benefit. Consider the previous example, where 'amazingly' is tokenised as `["amazing", "##ly"]`. By splitting the word like this, we automatically obtain information on both the meaning of the word (by the base 'amazing') and the fact that it is an adverb (by the suffix '-ly'). As a result, we can jointly learn the semantic meaning of many adverbs in the text at the same time, as most will include the same token for the '-ly' suffix.

2.1.2 Transformers

Because of the sequential nature of natural language, the first deep learning efforts in natural language understanding (NLU) made use of recurrent neural networks, such as Long Short-Term Memory networks (LSTMs) [14]. These recurrent models process the input sequentially, by keeping track of a hidden state h_t at time step t, and using that to process token x_{t+1} from the input sequence and produce h_{t+1}. The hidden state captures the information of all previous tokens in the input, meaning the sequential nature of the input text is conserved. However, this method of sequential processing makes it impossible to train these models in a parallel manner, something which has become increasingly important as sequences, model architectures and datasets kept growing.

Another way to process sequential data is to use convolutional neural networks. In these models, the network learns a set of convolutional filters of a

fixed window size, that slide over the input sequence. Although this method does allow for parallelised training, dependencies over spans longer than the window size can no longer be captured in a single layer of the model. As a result, learning long-distance dependencies in the text becomes a more difficult problem, which needs to be addressed by stacking several convolutional layers on top of each other.

In 2017, Vaswani et al. proposed the Transformer architecture as an efficient and effective architecture for dealing with sequences of text [33]. The Transformer architecture provides the basis for the majority of applications using state-of-the-art neural language models.

Overview

An overview of the architecture of the Transformer model can be found in Figure 2.1. Like many established architectures before it, the Transformer is a sequence-to-sequence model consisting of an encoder stack and a decoder stack. The encoder takes an input sequence and transforms it into an encoding. The decoder takes the encoding from the encoder and the start of an output sequence and produces the next token in the output. By starting with an empty sequence, and iteratively decoding a single token and appending it to the output sequence, the model can generate full sequences as output. Hence, the name: "sequence-to-sequence".

Multi-head attention

At the heart of the Transformer architecture lies the *attention* mechanism. Specifically, the Transformer implements scaled dot-product attention, which is also illustrated in Figure 2.2a. An attention layer receives a list of queries Q, keys K and values V. The output is a weighted average of each of the values in V, where the weights are computed based on the queries and keys. Specifically, the attention is computed as:

$$\text{Attention}(Q, K, V) = W \times V$$

with

$$W = \text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right),$$

where d_k is the dimension of the keys and queries. The scaling factor $\frac{1}{\sqrt{d_k}}$ is included to limit the problem of vanishing gradients in the softmax function.

In the Transformer architecture, *self-attention* plays an important role. Self-attention is described as attention in which the queries, keys and values all come from the same place. In the encoder, for instance, self-attention is used to ensure each token in the input sequence can *attend to* each other token in the input sequence. This allows the attention layer to model relationships

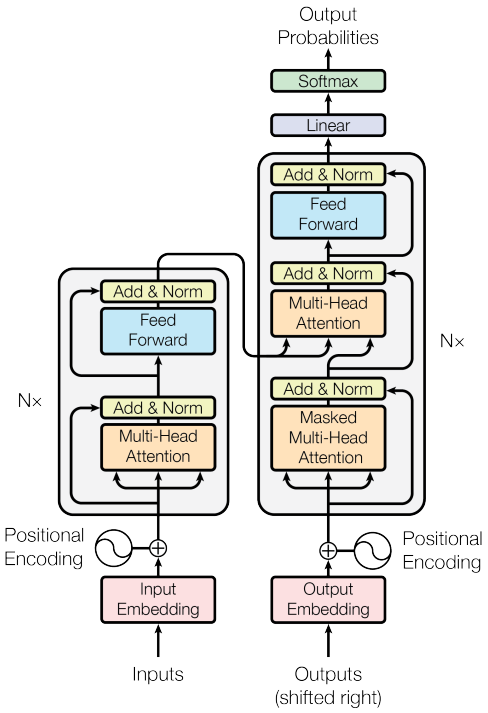
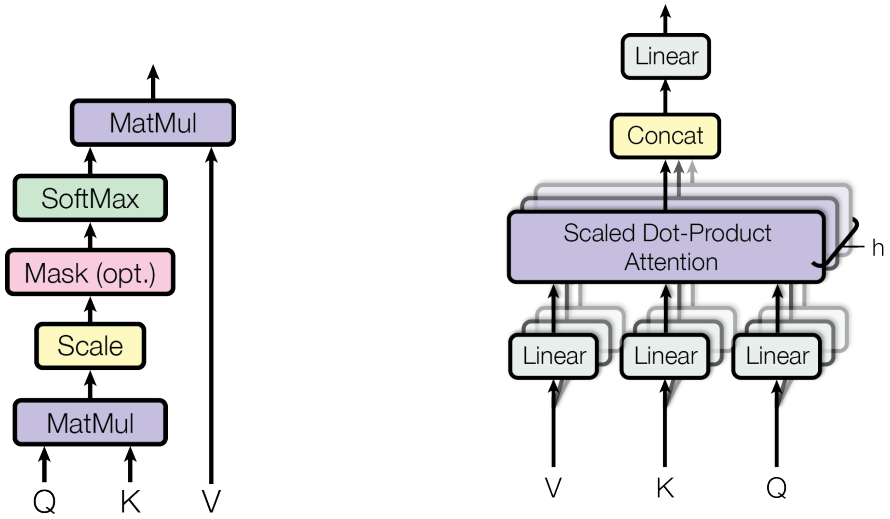


Figure 2.1: A general overview of the Transformer architecture, with an encoder stack (left) and a decoder stack (right). Image taken from [33].



(a) Scaled dot-product attention.

(b) Multi-head attention.

Figure 2.2: Attention mechanisms in the Transformer architecture. Images taken from [33].

and dependencies between different tokens in the input sequence. Since self-attention spans the entire input sequence, dependencies between any two tokens can be modelled in a single attention layer, as opposed to the multiple layers required for a convolutional model.

While the self-attention mechanism can efficiently model dependencies between tokens, the weighted averaging in the output makes it difficult to model multiple dependencies for a single token in the input sequence. For instance, take the following sentence: "John said he was feeling unwell". For the token 'he', there is an obvious dependency with the token 'John'. However, to correctly model the full meaning of the sentence, the model should also capture a dependency between 'he' and 'unwell'. With a single attention layer, the model only learns an averaged dependency on both parts of the input sequence. To overcome this issue, the Transformer uses *multi-head attention* instead. See Figure 2.2b for an illustrated overview of the multi-head attention mechanism.

With multi-head attention using h heads, the queries, keys and values are projected into h different representations each. For each combination Q_i, K_i and V_i, we compute $\text{head}_i = \text{Attention}(Q_i, K_i, V_i)$. Then, the outputs of all attention heads are concatenated and projected back into d_{model} dimensions. Formally:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) \times W^O$$

with

$$\text{head}_i = \text{Attention}(Q \times W_i^Q, K \times W_i^K, V \times W_i^V),$$

where W_i^Q, W_i^K, W_i^V and W^O are parameter matrices that are learned during training.

In the case of self-attention, Q, K and V all contain the same values (albeit projected in different ways): either the encoded input sequences (e.g. the one-hot encoded vector representation of `["j", "##oh", "##n", "said", "he", "was", "feeling", "un", "##well"]`), or the embedding outputs of the previous transformer layer.

In Figure 2.1, we see that the encoder consists of N stacked encoder blocks. Each encoder block consists of a multi-head self-attention layer, followed by a simple position-wise feed-forward network. Both the attention layer and feed-forward network are combined with residual connections, meaning the output of each sub-block is summed up with its input, after which the result is normalised.

The decoder consists of N stacked decoder blocks, which also contain a multi-head self-attention layer and position-wise feed-forward network. However, since the sequence the decoder receives as input is generated iteratively, tokens in the output sequence should only be allowed to attend to tokens that occur earlier in the sequence. Hence, the self-attention layer in the decoder blocks is *masked*, to ensure this restricted attention is enforced. Also, the decoder block somehow needs to incorporate the output of the encoder into its output. To do so, the decoder block contains an extra multi-head attention layer in between the other two layers, which obtains its keys and values from the encoder's output, and its queries from the preceding self-attention layer. And finally, the decoder ends with a linear layer with a softmax activation, to output token-level probabilities.

Positional encoding

Unlike the recurrent and convolutional models, the Transformer's attention mechanism does not inherently take the order of the inputs into account. To account for the fact that natural language is an ordered sequence of tokens, the Transformer adds a positional encoding to the input sequence. For each token, which is represented by a vector of dimension d, the positional encoding consists of two interleaved sinusoids, one a sine and one a cosine. Formally, for a token at position pos, the positional encoding is vector PE of length d, with, for $0 \leq i < d/2$:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

By choosing sinusoids to represent the positional information, the model should also be able to handle sequences longer than the ones seen during training time.

2.1.3 BERT

To train a complex and large model such as the Transformer, you need access to a large amount of training data. Since labelled data for supervised objectives is generally costly to obtain and not available in abundance, most models nowadays make use of *unsupervised pre-training*. With pre-training, the model is first trained on a large corpus of textual content without labels, where the goal is to gain a general understanding of the natural language used in the documents. After the model has been pre-trained sufficiently, it is fine-tuned on the labelled data of the supervised task.

Since pre-training is unsupervised, the pre-training objective should force the model to gain an understanding of natural language, without supplying external labels. Early pre-training objectives often included the task of predicting the next token in a sentence [7, 24]. However, with this objective, each token is only conditioned on the preceding tokens, meaning the model will never learn to incorporate the full sequence in the representation of a specific token. To allow for bi-directional text understanding, Devlin et al. introduced their model BERT (Bidirectional Encoder Representations from Transformers) with the *masked language modeling* (MLM) pre-training objective [9].

BERT is a standard Transformer encoder stack, without the decoder. It is pre-trained on the BooksCorpus [41] and English Wikipedia. The MLM objective used to pre-train BERT randomly selects 15% of the tokens in the input sequence. From these tokens, 80% will be replaced by a special `[MASK]` token, 10% will be replaced by a random token in the vocabulary, and 10% will be left unchanged. The model is then asked to reproduce the full input sequence. Since random parts of the input sequence are corrupted, the model is forced to capture the relationship between different parts of the sequence. Applications of BERT to NLP tasks have shown a basic level of semantics got encoded in the network's structure and parameters.

Besides the MLM objective, BERT is also pre-trained with the *next sentence prediction* (NSP) objective. For the NSP objective, BERT encodes two sentences A and B by joining them with a separator token `[SEP]` and prepending a special `[CLS]` token. The resulting sequence would thus be: `"[CLS] A [SEP] B"`. BERT also includes embeddings E_A and E_B to denote, for each token, whether they belong to sequence A or sequence B. The output of the encoder for the `[CLS]` token can be seen as the combined representation of the entire input sequence and is consequently used for classification of the

full sequence. The NSP objective then follows as a binary classification on A and B, where the output indicates whether B is the correct sentence to follow A.

Because of the MLM pre-training objective, BERT can effectively learn the meaning of a sentence and the dependencies between words within a sequence. The NSP objective, on the other hand, allows BERT to capture the relationship between different sentences altogether. By combining the two, BERT gains a reasonable understanding of natural language, which proves effective when fine-tuning it on a variety of downstream tasks.

2.2 Structured information extraction

Structured information extraction describes the general task of extracting the values of structured attributes from unstructured or semi-structured documents. Generally, the task is considered in two different settings: *closed* information extraction (ClosedIE) and *open* information extraction (OpenIE).

In the ClosedIE setting, the *schema* of the entity is considered to be known. In other words, the set of attributes that can be found for a specific entity is fixed and known before the extraction happens. The task is then to find, for every attribute in the schema, the corresponding value(s) in a specific document.

In the OpenIE setting, the set of attributes is not known. Hence, the task not only requires the model to extract attribute values, but also the corresponding relation. In general, OpenIE is defined as the extraction of triples $\langle s, r, o \rangle$ from webpages, where s is the subject entity, r is the relation type, and o is the object/value.

Early approaches for information extraction made use of *wrapper induction* to extract attributes from a web page. A wrapper defines an exact set of rules to select one or more specific elements from a webpage. Examples of such rules are the surrounding tokens for an attribute, such as `<b>` and `</b>` [19], or XPath queries [12]. While wrappers can be very precise, they do not generalise well to new websites and run the risk of breaking when the site structure is updated. They also require manual annotation of at least a couple of pages for each website, which is a labour-intensive process.

To reduce the amount of manual annotation necessary, several alternatives have been considered. For instance, WEIR [5] uses data redundancy in partially overlapping web sources to find relevant rules for a given page. Hao et al. [13] describe an approach where general features are learned for HTML DOM nodes, that translate well to unseen websites. CERES [20]

Table 2.1: Overview of the SWDE dataset.

Vertical	#Pages	Attributes
Auto	17,923	model, price, engine, fuel-economy
Book	20,000	title, author, ISBN-13, publisher, publish-date
Camera	5,258	model, price, manufacturer
Job	20,000	title, company, location, date
Movie	20,000	title, director, genre, MPAA-rating
NBA player	4,405	name, team, height, weight
Restaurant	20,000	name, address, phone, cuisine
University	16,705	name, phone, website, type

and OpenCeres [21] moved away from manual annotations altogether, and instead are trained using a domain-specific knowledge base.

More recent approaches try to learn general representations of HTML DOM trees, e.g. by generating node-level embeddings [40] or by representing the DOM tree using a context-free grammar [6]. Other approaches use pre-trained Transformer networks and apply varying methods to incorporate the HTML information into the network. For instance, [8] encodes HTML tags and attribute values into the sequence, and adds additional positional encodings to represent several tree-based features of the DOM node. [37] also adds the tag and certain attribute values into the sequence, but uses the positional encodings to incorporate visual 2D position information. The model in [34] contains additional attention mechanisms that are used to encode the DOM tree and learn the relationships between the DOM nodes and textual content.

2.2.1 Structured Web Data Extraction (SWDE) dataset

In order to test and evaluate ClosedIE extraction methods, the Structured Web Data Extraction (SWDE) dataset [13] was created. The SWDE dataset contains entity pages (i.e. pages with a single subject entity) for 8 different verticals, each with 10 websites. For each vertical, 3 to 5 attributes were picked, and all pages for the vertical were annotated for those specific attributes. An overview of the verticals and corresponding attributes can be found in Table 2.1.

Due to the limited set of attributes per vertical, the SWDE dataset is not particularly suited for the evaluation of OpenIE models. For that reason, an expanded version of the dataset was created for the 'movie', 'NBA player' and 'university' domains [21]. The Expanded SWDE dataset contains all the

relations found on each specific page, instead of being limited to at most 5 relations per vertical. This allows it to be used to evaluate OpenIE systems.

The SWDE dataset has been around for over a decade and is established in the information extraction literature as the standard dataset for evaluating information extraction systems. This makes it a perfect dataset to evaluate our models and compare them against the existing literature.

2.2.2 Evaluation metrics

Global metrics

Evaluating an information extraction system can be done by viewing the extraction problem as a classification over all triples $\langle s, p, o \rangle$. An extracted triple is then correct if all of s, p and o match the corresponding values of a ground truth triple. Using this classification approach, we can compute precision, recall and F_1 scores over the entire evaluation set. This evaluation method can be used for both ClosedIE and OpenIE systems.

To account for slight, meaningless differences between the extracted values and ground truth values, we normalise both sets of values by removing punctuation and double whitespace. We define the sets of triples $\mathcal{R}_d^{true}$ and $\mathcal{R}_d^{pred}$ for ground truth and predicted relations of document d, respectively:

$$\begin{aligned} \hat{y}_{d,a} &= \textsc{Normalise}(\mathrm{GT}(d, a)) \\ y_{d,a} &= \textsc{Normalise}(\textsc{Extract}(d, a)) \end{aligned}$$

$$\begin{aligned} \mathcal{R}_d^{true} &= \{\ \langle d, a, \hat{y}_{d,a} \rangle \mid a \in \textsc{Attrs}(d)\ \} \\ \mathcal{R}_d^{pred} &= \{\ \langle d, a, y_{d,a} \rangle \mid a \in \textsc{Attrs}(d)\ \}, \end{aligned}$$

where $\textsc{Attrs}(d)$ is the set of ClosedIE attributes for document d, Normalise is the normalisation procedure introduced above, $\mathrm{GT}(d, a)$ is the ground truth value of attribute a for document d, and $\textsc{Extract}(d, a)$ is our model's prediction for the value of attribute a for document d.

Using these definitions, we formally define the global metrics as:

$$\begin{aligned} P_d &= \frac{|\mathcal{R}_d^{pred} \cap \mathcal{R}_d^{true}|}{|\mathcal{R}_d^{pred}|} \\ R_d &= \frac{|\mathcal{R}_d^{pred} \cap \mathcal{R}_d^{true}|}{|\mathcal{R}_d^{true}|} \\ F_{1,d} &= 2 \cdot \frac{P_d \cdot R_d}{P_d + R_d} \end{aligned}$$

To obtain aggregated metrics over the entire dataset, we macro-average the metrics over all documents.

$$P = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} P_d$$
$$R = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} R_d$$
$$F_1 = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} F_{1,d},$$

where $\mathcal{D}$ is the set of documents.

When performing statistical significance tests for the global metrics, we consider P_d, R_d and $F_{1,d}$ to be random variables. We then compare the (variation between the) distributions of these variables for two different models, and perform a paired Wilcoxon signed-rank test to determine, per metric, whether one of the two models performs significantly better than the other. For each of our significance tests, we use a significance level of $\alpha = 0.01$.

Instance-level metrics

A limitation of measuring exact triple matches is that it disregards the cases where an extracted attribute is partially correct. For instance, a predicted date of 'January 3rd' does not exactly match the ground truth of 'January 3rd, 2019'. However, it is more correct than other values like 'July 30th', and we would like for that to be reflected in the evaluation metrics. To do so, we can borrow metrics from the question answering domain [26].

In question answering, the goal is to extract an answer to a question from a given context. Evaluation is done on a per-question basis. Again, both the predicted answer and the ground truth answer(s) are normalised, i.e. double whitespace, punctuation and articles (a/an/the) are removed. With these normalised values, two metrics are defined. The *exact match* (EM) metric measures whether the prediction fully matches one of the ground truth answers and, as such, disregards partial matches. As such, it can also be described as the accuracy over the normalised values. The F_1 metric treats each answer as a bag of words, and computes the F_1 score between the prediction and each of the ground truth answers. As a result, the F_1 metric captures overlap between two answers and rewards partial matches, though it disregards the order of the tokens in the answers.

The EM and F_1 metrics are computed for every question and are averaged over all questions in the evaluation set to obtain a final score. In the extraction task, we consider each combination of webpage and attribute to be a question, and the attribute value to be the answer. Then, we average over all web pages and attributes to obtain the evaluation score.

These question answering metrics have been used for the evaluation of information extraction models in [34], where the authors have kept the names of the metrics consistent with their names in the question answering domain: EM and F_1. However, to avoid confusion between the previously introduced classification F_1 score and this instance-level F_1 score, we have opted to use a different name for the second metric: *weak match*, or WM.

Formally, we can define the EM and WM metrics for each document d and attribute $a \in \textsc{Attrs}(d)$ as follows:

$$EM_{d,a} = \begin{cases} 1 & \text{if } \hat{y}_{d,a} = y_{d,a} \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} P_{d,a} &= \frac{|\textsc{Tokens}(y_{d,a}) \cap \textsc{Tokens}(\hat{y}_{d,a})|}{|\textsc{Tokens}(y_{d,a})|} \\ R_{d,a} &= \frac{|\textsc{Tokens}(y_{d,a}) \cap \textsc{Tokens}(\hat{y}_{d,a})|}{|\textsc{Tokens}(\hat{y}_{d,a})|} \\ WM_{d,a} &= 2 \cdot \frac{P_{d,a} \cdot R_{d,a}}{P_{d,a} + R_{d,a}}, \end{aligned}$$

where Tokens converts a value to a bag of words by splitting the value on whitespace. The performance over the entire evaluation set is computed by averaging the EM and WM scores over all queries.

$$\begin{aligned} N_Q &= \sum_{d \in \mathcal{D}} |\textsc{Attrs}(d)| \\ EM &= \frac{1}{N_Q} \sum_{d \in \mathcal{D}, a \in \textsc{Attrs}(d)} EM_{d,a} \\ WM &= \frac{1}{N_Q} \sum_{d \in \mathcal{D}, a \in \textsc{Attrs}(d)} WM_{d,a} \end{aligned}$$

To determine whether one of two models performs significantly better than the other, we again consider $EM_{d,a}$ and $WM_{d,a}$ to be random variables and perform a paired Wilcoxon signed-rank test (with $\alpha = 0.01$) to determine whether the variation between the models is statistically significant.

Note that this set of metrics can only be used for the ClosedIE setting, as it requires us to have a concrete set of attributes per webpage. In the OpenIE setting, we try to extract all possible attributes, making it impossible to frame the task as a question-answering problem.

Throughout the remainder of this thesis, we will refer to the first set of metrics (P, R and F_1) as *global* metrics, while we refer to the second set of metrics (EM and WM) as *instance-level* metrics.

Chapter 3

Research objectives

3.1 Motivation

While the Web contains vast amounts of information, large-scale automated systems often require data in a particular, structured form. For instance, search engines can use knowledge of structured data on websites to enrich search results with infoboxes and question-answering modules (or "entity cards"). Likewise, virtual assistants can assist much better with tasks if they can understand the requirements of the task and have knowledge of other task-specific attributes.

Recognising which parts of a webpage are significant and correspond to which type of information is a task that is trivial for humans to perform. For instance, when opening a page with a recipe, one immediately notices the title of the recipe, the required ingredients and the instructional steps. This insight comes from both the fact that people can understand the textual contents of the webpage instantly and relate that to their common knowledge, but also from the expectations one has of the layout of the page. A title of a recipe is usually found at the start of the page, and marked as a heading with larger font size, bold text and possibly a different colour.

For machines, this distinction is a lot harder to make. They can read the HTML source code, parse the layout information from it, and render the page so it is viewable by people. However, webpages are only *semi-structured*. Of course, HTML introduces structure in the layout of the page. However, the actual contents of the page still mostly consist of natural language, which makes it difficult for machines to understand what structured data is represented on the page - especially because computers lack the natural language understanding capabilities and common knowledge that humans possess.

3.2 Opportunities

Recent advances in neural language models [33, 9, 25] have significantly improved the language understanding capabilities of computer systems. By combining these models with the structural information that HTML gives us, we could come closer to mimicking the insights that allow people to intuitively recognise and understand structured data on web pages.

Lack of training data is usually a big issue with the usage of supervised models. Especially if we want to train large, complex language models, we need access to a large amount of labelled data. While several datasets exist for the information extraction task, they are merely useful for evaluation purposes - they are not large or varied enough to enable training for models that can extract many different types of data from many different websites.

In recent years, a lot of effort has been put into making structured data more accessible to automated systems like search engines [11, 22]. By incorporating structured data annotations in their pages and adhering to a pre-defined schema (such as Schema.org [11]), website owners have made it increasingly more feasible for machines to understand what structured data is present on the page, and informed them about the relevant attribute values. While this has already become a useful data source for search engines and other parties, these annotations are still missing from more than half of the internet [1]. Conversely, the annotations that we have, on the other half of the internet, can double as a source of training data. By using the annotations provided on certain websites, we can train a model that learns what the data for a specific schema type looks like. If general and accurate enough, this model can then serve as a way to extract information from other websites that do contain structured data but lack the annotations.

3.3 Challenges

Given the success with transformer model models throughout NLP tasks, we would naturally consider a transformer-based model for structured information extraction as well. Although they are very powerful, transformer-based neural language models suffer from size limitations. Since the attention mechanism's memory requirements grow quadratically with the length of the input sequence, many of these models only support sequences with a certain maximum length. For example, BERT and T5 use a maximum input sequence of 512 tokens. Webpages are generally longer documents, that do not fit in such limited sequence lengths. Thus, to avoid prohibitively expensive resource requirements, we need a way to circumvent these issues, either by splitting the input into separate subsequences or by using a different attention mechanism.

Since different websites on the Web are created by different people and consist of different layouts and contents, there is a lot of variety in the HTML structure between different pages. This means that a model should not just memorise the structure of the documents in the training set. Instead, if it is to work across many different pages on the internet, it needs to learn more general relations between DOM nodes and their contents.

In the task of information extraction, the type of attribute that is being extracted can impact the effectiveness of a model by a large margin. For instance, Foley et al. [10] showed that more regular fields like dates and locations were more easily recognisable than attributes that are longer and more varied, such as titles or descriptions. For a general approach that is effective for many types of structured data, our models need to be able to handle these different types and representations of attributes accurately.

3.4 Research questions and scope

The ultimate goal of this thesis is to propose a system that can extract structured data from HTML pages. We will explore in which way neural language models can help achieve this objective. We also investigate the role of HTML structure and consider ways in which to incorporate the structural information given by the HTML in our model. These objectives are studied in terms of the following research questions:

RQ I. How can we leverage powerful neural language models to enable effective attribute extraction?

- a) Are encoder-only or sequence-to-sequence language models more effective for the information extraction task?

RQ II. How does the size of the HTML segment influence the performance of the model?

RQ III. How can we encode the structure of the HTML segment into sequences of text in a meaningful and effective way?

RQ IV. How well do our models generalise to new, unseen websites?

Given the context of using structured data annotations from Schema.org as training data for our models, it makes sense to assume that we have access to a schema for our data types. As a result, we will only be considering the ClosedIE setting in this research. Whether the developed methods also prove useful and effective for the OpenIE domain has to be considered in future research.

In practice, many structured data types will have multi-valued attributes (such as ingredients for a recipe, or actors in a movie). However, we decided to limit the scope of our research to the extraction of a single value per attribute. The reason for doing this is two-fold. First, the extraction of single-value attributes is a simpler problem than the multi-value case, which allows us to study the problem in a more general setting. Second, the current state-of-the-art ClosedIE systems all report their findings for the single-value case. By assuming a similar setup, we ensure our approach is comparable to the other systems.

Similarly, the scope of our research is limited to the extraction of attributes on entity pages, i.e. web documents that only discuss a single entity. This limitation follows from the choice of using the SWDE dataset (Section 2.2.1) for evaluation purposes.

Chapter 4

Methods

4.1 Design

A general overview of our model can be found in Figure 4.1.

4.1.1 Segmentation

As mentioned in Section 3.3, HTML documents are unlikely to fit in the maximum sequence length of 512 tokens that is maintained by many neural language models. One way to circumvent this problem is to truncate the documents up to the first 512 tokens. However, this approach clearly breaks the model's ability to extract data from later parts of the page. Other approaches [2, 39, 3] have been introduced that do not employ (quadratic) full self-attention, but rather use a combination of windowed and global attention to make the complexity of the attention mechanism linear in terms of the length of the input sequence.

A way to still be able to use full self-attention but also handle longer documents is to split the document into different segments. Each segment can then be processed separately, and the outputs for each segment can be aggregated to obtain the final output for the full document. An issue with the segmentation approach, though, is that you might split highly contextually dependent parts of the sequence into separate segments. In the information extraction task, you might split an attribute from the label that announces it or even split the value itself in half. To overcome this, you could use a sliding window, where segments are partially overlapping. This ensures for each feature of interest that there is always at least one segment that contains the full context necessary to extract the value.

Since we are working with semi-structured HTML data, we opted for a different approach. The layout of a document is defined by the DOM tree,

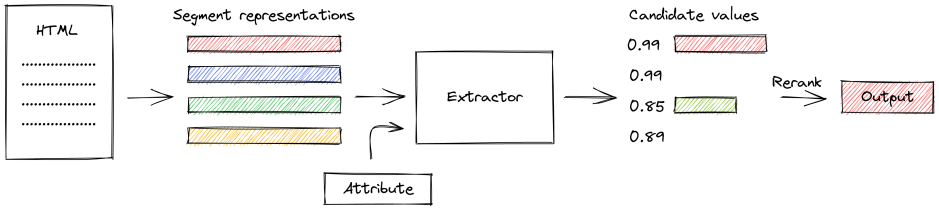


Figure 4.1: An overview of our proposed model. Documents are first split into segments, and each segment is represented as a sequence of text. Each segment is fed into an information extraction model, and the results for segments are ranked to output the final prediction for a given document.

meaning nodes that are contextually dependent on one another are more than likely contained in the same DOM subtree. Instead of segmenting the document on a token basis, we can thus segment the page by taking different subtrees of the DOM tree. At which depth we cut off a specific subtree depends on the size of the context that we want to take into account. If we go down deeper into the tree structure, we will have less context to work with, and the models are likely to learn more local features. If we segment higher up in the tree, our context will be larger and our models can pick up more global features, at the cost of a higher memory requirement.

To obtain segments of a page that fit within a certain context length c, we simply traverse the DOM tree. For every node n, we try to represent that node and its children with the representation R_n we introduce in Section 4.1.2. If it fits within the context size c, we produce R_n as a representation for the entire subtree that has n as its root. If not, we traverse the tree further down and try to segment each of n's children individually. This approach ensures we find segments with a maximum context size while ensuring their representations are sequences of at most c tokens. Our segmentation procedure is also illustrated in Figure 4.2.

In order to see the effect different context sizes have on the performance of our models (RQ II), we experimented with maximum context sizes of 128, 256 and 512 tokens.

4.1.2 HTML representation

In Section 3.2, we mentioned that the structure of the HTML document can be very useful in understanding what type of data is represented on the page. Not only can we use the layout of nodes in the tree structure, the types of the nodes (e.g. `div` or `span`), and their attributes (e.g. `id` or `class`) can also be highly informative. As a result, we would like to incorporate these features of the HTML document into our input sequences.

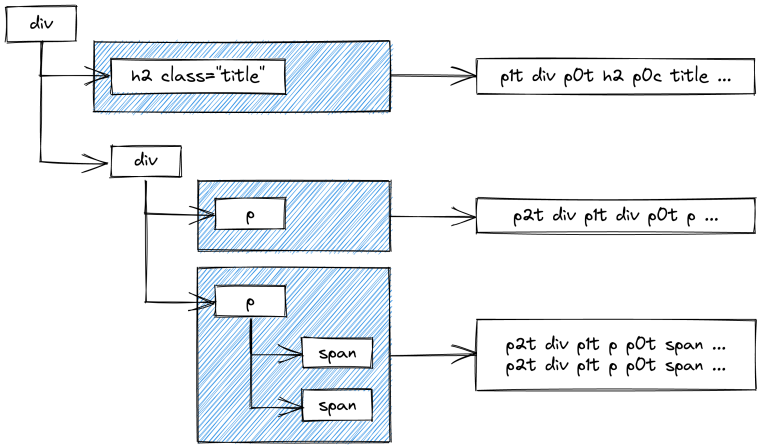


Figure 4.2: Our tree-based segmentation procedure. The tree is traversed until a subtree can be represented in the specified sequence length. The top-level `div` and the second `div` cannot be represented in C tokens, so they are split into multiple segments.

```
<div class="container">
    <h1>This is a heading</h1>
    <p id="description">
    This is the description.
    </p>
</div>
```

Listing 4.1: An example HTML fragment.

A simple method for encoding the HTML document in a sequence is to traverse the tree in-order and insert the HTML tags into their corresponding location in the sequence. An example encoding of the HTML fragment in Listing 4.1 would be:

```
<div> class=container <h1> This is a heading <p> id=description
↪  This is the description.
```

By separating the attributes from the HTML tags and adding the tags as special tokens to the tokeniser, the model can define a clear meaning for each of the different tags. Although this approach seems attractive due to its simplicity, it does require the model to 'reconstruct' the input tree based on this representation - a task that could be significantly harder for the model to learn than understanding the semantics of a natural language text.

Instead, our representation is still created by traversing the DOM tree and producing the text nodes in-order. We do not include the HTML tags in their corresponding points in the sequence, though, but we prefix every text

node with the representation of their a most recent ancestors. This ensures that the tree structure is embedded into the sequence, at the cost of requiring a longer sequence to represent the same subtree and only including the information of a fixed amount of ancestors.

Each parent element is represented as a combination of its tag name, and (if present) its `id` and `class` attributes. We represent each such value with the sequence `p[i][t] [value]`, where `[i]` is the ancestor level (i.e. the distance from that node to the current text node) and `[t]` represents the type of the value (`t`, `i` or `c` for tag name, id and class, respectively). The fragment in Listing 4.1 would then be represented as the following sequence:

```
p1t div p1c container p0t h1 This is a heading p1t div p1c
↪  container p0t p p0i description This is the description.
```

We hypothesise the following features of this representation:

- Representing the a most recent ancestors before each text node allows the model to understand relationships between the ancestors of a node and the contents of that node more easily. This is because the hierarchy of HTML nodes is made explicit in the encoded sequence, and the model does not have to learn complex tree-like relationships itself.
- By representing the tags as `p[i]t [tagname]`, we do not force the model to learn the semantic meaning of each individual HTML tag (such as `<div>` and `<p>`). Instead, we allow it to learn that `p[i]t` represents a HTML tag, and `[tagname]` supplements this meaning by giving it more information on the type of HTML tag. This allows the model to obtain a more general understanding of the HTML structure.
- Since the ancestor index `[i]` is a number, and the surrounding characters are letters, the tokeniser will almost be guaranteed to split this information triplet into distinct tokens. As a result, it should be able to learn the meaning of each aspect of the encoding separately. In other words, it might learn that `p` represents an HTML node, `[i]` represents how distant or relevant that node is to the current text node, and that `t`, `i` and `c` represent different types of information of that specific ancestor.

Since the encoding of ancestors takes up quite a big portion of the resulting sequence, the size of the context a segment can capture decreases as we incorporate more HTML information. As a result, we need to make a trade-off between the level of detail with which we represent ancestors on the one hand, and the size of the subtrees we want to be able to fit in a single segment on the other hand. To find a representation that balances these requirements (thus answering RQ III), we performed an experiment investigating the performance of our models with different HTML representations.

4.1.3 Extraction versus generation

When working with large language models, a choice must be made between using an encoder-only model or an encoder-decoder (i.e. sequence-to-sequence) model. As far as we are aware, no existing literature touches upon the benefits of one over the other in the setting of information extraction. However, the setup of our task is similar to the task of question answering [26], where the model is provided with a query and a context, and the goal is to find the span of text in the context that best matches the query. Hence, we consider the usage of both types of models in the question-answering domain and hypothesise how their advantages and disadvantages might extend to our research.

Encoder-only models like BERT [9] and SpanBERT [17] are used by appending two classification heads on top of the encoder, one that predicts the token at the start of the span containing the answer, and one that predicts the end of the span. The span that has the highest joint probability of start and end position is produced as the predicted answer. The null answer (i.e. the prediction that the context does not contain the answer) is produced by predicting the `[CLS]` token as both the start and end token. Encoder-decoder models produce the output by taking the encoded query and input sequence and using autoregressive decoding to generate the expected answer. The null answer is produced by generating the empty sequence.

A major advantage of the encoder-only models is that the span selection procedure guarantees that the predicted answer is a part of the provided context. The lack of a decoder makes the model easier to train, and inference is faster as no auto-regressive decoding is required. However, the training procedure of an encoder-only model is a bit more involved, as it needs ground-truth labels on the exact start and end positions of each answer in the training set.

In recent years, sequence-to-sequence models like T5 [25] have been shown to outperform the encoder-only models in the question-answering task SQuAD [26]. The fact that the sequence-to-sequence framework allows for a unified approach in handling a variety of text-based problems makes it an attractive solution for pre-training and fine-tuning on many tasks. In fact, the published T5 models are already pre-trained on the SQuAD task, which could prove beneficial as the information extraction task can be defined similarly. Additionally, the fact that the output is generated means we can directly use the ground-truth labels as training targets, instead of having to locate these in the input sequence to assign the corresponding start and end positions. A disadvantage of the sequence-to-sequence model is that purely generating the answer does not guarantee that the predicted attribute is a part of the input sequence, and the model might even hallucinate and produce nonsensical

outputs. To alleviate this issue, one might make use of constrained auto-regressive decoding [29] or copy mechanisms [38]. In our experiments with sequence-to-sequence models, we opted for the former (see Section 4.2.2).

Since neither of the architectures is inherently better than the other, we experimented with both setups to research empirically which of the architectures works best for our problem (thus answering RQ I.a).

As a final note, the tree-like structure of the HTML documents also allows us to frame the information extraction task as node classification (or 'node tagging'), as is implemented in [8, 40]. Instead of selecting or generating the appropriate value from the text, we could assume that a specific DOM node (or subtree) exists whose textual content is exactly equal to the attribute value. Under that assumption, the task can be reduced to the selection of the node that contains the attribute value. While this sounds like a reasonable hypothesis, we decided to stick with the more general approach of extracting a span of text from the document. We did not carry out further experiments to test whether the assumption is valid, nor did we implement a node tagging variant of our model. We believe this could be picked up in future research, though (see Section 7.2).

4.1.4 Segment aggregation

As mentioned in Section 4.1.1, each document is segmented into a number of different, non-overlapping sequences. For each of the sequences, we try to extract the requested attribute. The resulting prediction is a (possibly empty) span of text, and a corresponding confidence score. The confidence is computed in different ways for the encoder-only and the encoder-decoder models, which we will discuss in more detail in Section 4.2.2.

To produce a final document-level prediction for an attribute, we need to somehow aggregate or rank the predictions made for each of the segments. We do this using a very basic approach:

- If *any* of the segments contain a non-empty prediction, we select the non-empty prediction with the highest confidence.
- If all predictions for each of the segments are empty, we conclude that the attribute was not found.

Especially note the requirement that all segments must produce an empty prediction before we can conclude that the attribute was not found. Otherwise, we might accidentally produce the empty output, even though the attribute was present on the page. Consider the case where we split the page into two segments. The first segment contains only gibberish and very clearly does not contain a valid value for the attribute. Our model will produce the empty output with very high confidence. The second segment does contain

the attribute, but the model is less certain than for the previous case - e.g. because the attribute is difficult to recognise. As a result, the confidence for the correct prediction would be lower than the confidence for the empty prediction, and the model would conclude that the requested attribute is not present on the page.

In exploratory experiments, we have tested the usefulness of including a confidence threshold. Instead of outputting values with (too) low confidence scores, the model would decide that the attribute was not found on the page. This turned out to rather hurt than increase the model's performance, though, and we have refrained from using a confidence threshold in the remainder of our experiments.

As each segment is processed independently, the models only consider local context, and the ranking is based on local confidence scores. To incorporate more of the global context into the models, one could consider using additional information on the position of each segment in the webpage. For instance, one could render the input pages with a web browser and extract the 2D positions and sizes from the rendered page (similar to [37]). This would also allow the incorporation of additional features like font size or colour. However, rendering each webpage before extracting information incurs a high cost, and the rendering process could not easily be changed without breaking the compatibility between the training and inference data. Additionally, crawled webpages used for information extraction might rely on outdated and no longer existing resources (like stylesheets), which makes this approach suboptimal in practice.

An alternative and more efficient solution we considered is to use the index of each segment as a weak signal for their position on the page. Since the DOM tree is traversed in-order, this index contains some information on the (relative) position of a segment on the page. Preliminary analyses showed that attributes are often clustered close to each other in specific segments, usually at the start of the page. However, experiments leveraging this insight did not provide any increase in performance, and we had to discontinue this approach due to time constraints.

Aside from using positional information to influence the predictions and confidence scores of the model, we could also improve the ranking functionality of the model. In an approach similar to Fusion-in-Decoder [16], we could encode each of the segments separately, and select the best scoring candidates to pass on to another reranker or decoder. Our initial attempts at incorporating a separate reranker for the top candidates did not yield promising results, though, and we again had to abandon the approach because of time constraints. However, we still see this as a viable direction for future work.

4.2 Implementation

In this section, we discuss the specifics of the implementation of our system. The code used for our experiments can be found on GitHub[1]. Our experiment dashboard will be published on Weights & Biases[2].

4.2.1 Preprocessing

To parse the HTML documents and traverse the HTML tree, we used the `lxml` library[3]. The pseudocode for the segmentation procedure can be found in Algorithm 1.

For each node we traverse, we encode it in the representation introduced in Section 4.1.2 using the REPRESENTATION method. If the representation was empty we return the empty set, as no segments could be found. If the representation is larger than the sequence length limit C, we traverse each of the node's children and extract all segments from the corresponding subtrees. Note that setting C to be too small might cause large leaf nodes to be skipped, as their content is too large to represent in a single segment but they cannot be split into further subtrees.

If the representation does fit in the desired sequence length, we take it as a representation for a single segment. For each segment, we also need to keep track of which attribute values it contains. To do so, we normalise all attribute values and check whether these normalised values are contained in the segment's textual representation (i.e. the representation without any HTML encodings). The NORMALISE method is the same normalisation as the one used in our metrics (see Section 2.2.2), meaning it transforms the text to lowercase, strips whitespace and punctuation and removes the articles 'a', 'an' and 'the'. When searching for the attribute value in the segment's text, we want to ensure that the attribute values are only found if they are separate words, and not when they are contained in other words. For instance, we do not want the MPAA rating 'R' to be matched with every sequence that contains a single letter 'r'. Instead, the CONTAINS method creates a regular expression of the value by surrounding it with `\b` (word boundary) escape characters. By pattern matching with this regular expression, we ensure the attributes that we find always contain full tokens.

Every attribute can have zero or more values associated with it for a specific document. For each attribute, we keep track of all of the values of that attribute that are contained in the segment text. For every combination of these values (i.e. every tuple in the Cartesian product of the value sets), we produce a separate segment with the found HTML representation and the

[1] `https://github.com/gijshendriksen/master-thesis/`
[2] `https://wandb.ai/gijshendriksen/information_extraction`
[3] `https://lxml.de`

Algorithm 1 Our segmentation algorithm SEGMENT(n, C, A)

Require: n, an `lxml` HTML node
Require: $C \geq 0$, the maximum length of each segment
Require: A, the ground truth attribute values for the current document
Ensure: n is split into a set of segment representations S with each $|S| \leq C$

```
R ← REPRESENTATION(n)
if |R| = 0 then
    return ∅
else if |R| > C then
    Segments ← ∅
    for child ∈ CHILDREN(n) do
        Segments ← Segments ∪ SEGMENT(child, C, A)
    end for
    return Segments
else
    Segment ← ASSOCIATIVEARRAY({'text' ⇒ R})
    Segments ← {Segment}
    Text ← NORMALISE(TEXTREPRESENTATION(n))

    for (attr, values) ∈ A do
        values ← {val | val ∈ values, CONTAINS(Text, NORMALISE(VAL))}
        if EMPTY(values) then
            Segments ← {S [attr ⇒ null] | S ∈ Segments}
        else
            Segments ← {S[attr ⇒ val] | S ∈ Segments, val ∈ values}
        end if
    end for
    return Segments
end if
```

corresponding set of values. To illustrate, suppose we have a single segment with representation R and attributes $attr_1, attr_2$ and $attr_3$. If both $attr_1$ and $attr_2$ have two values present in this segment, and $attr_3$ has none, we obtain four different segments:

$$\{'text' \Rightarrow R, attr_1 \Rightarrow A_1, attr_2 \Rightarrow B_1, attr_3 \Rightarrow \textbf{null}\}$$
$$\{'text' \Rightarrow R, attr_1 \Rightarrow A_1, attr_2 \Rightarrow B_2, attr_3 \Rightarrow \textbf{null}\}$$
$$\{'text' \Rightarrow R, attr_1 \Rightarrow A_2, attr_2 \Rightarrow B_1, attr_3 \Rightarrow \textbf{null}\}$$
$$\{'text' \Rightarrow R, attr_1 \Rightarrow A_2, attr_2 \Rightarrow B_2, attr_3 \Rightarrow \textbf{null}\}$$

By not limiting ourselves to a single attribute value per segment, our models should be able to better recognise all different types of values for a given attribute. This should in turn improve the performance of the model, as it will have a more general understanding of what the attribute represents.

We note that our method of annotating each segment like this can lead to ambiguous annotations. For instance, if a book description contains mentions of the book's author, our annotation method will detect that the author is included in the segment containing the description, even though there are other parts of the page where the author is mentioned more recognisably. Likewise, a page about an NBA player might also contain a list of all NBA teams, and our annotation method will recognise the player's team in that list. As a result, our models will be trained to not only predict attribute values in their most obvious place on the page, but also on all other mentions throughout the page. This might lead them to focus more on the meaning and format of the attribute, rather than the HTML structure indicating that the attribute can be found there. However, we argue that the more recognisable attribute mentions on a page should be picked up by the model with a higher confidence, causing it to prefer the correct value over other possibly correct values. Also, it is non-trivial to only annotate the segments that specifically state the attribute value and ignore other segments that randomly mention the attribute value. Our annotation method ensures we always pick up all attribute values on a page, without making assumptions about which annotations to keep and which to discard.

We set up our preprocessing pipeline in such a way that we only have to extract the segments and generate the representations once. For each website in each vertical, we generate a single CSV file containing all the segments for that website. To allow experiments with different context sizes, we preprocess the data for the context sizes $C \in \{128, 256, 512\}$. We implemented multi-processing to speed up the pipeline by handling multiple web pages in parallel.

4.2.2 Language models

Our models are implemented using PyTorch [23], and we used the Hugging Face [35] library for pre-trained language models and tokenisers.

Encoder-only

For the encoder-only setup we use the `BertForQuestionAnswering` model and the `BertTokenizerFast` tokeniser. This model implements a standard BERT [9] model followed by two classification heads: one for the start position of the span and one for the end position. We initialise the model and tokeniser from the `SpanBERT/spanbert-base-cased`[4] repository, which provides the pre-trained models of SpanBERT [17]. SpanBERT is a variant of BERT that was pre-trained with an added objective of predicting missing spans of text from a sequence, a setting particularly well-suited for span extraction tasks.

To extract an attribute from a segment, we encode the inputs using the default method in the question-answering domain: we concatenate the attribute name and the segment representation, separated by the separation token. Given an `<attribute>` and a `<segment representation>`, we thus use the following sequence:

```
[CLS] <attribute> [SEP] <segment representation> [EOS]
```

The BERT tokeniser automatically assigns the token type 0 to the tokens that make up the attribute name and the token type 1 to the tokens in the segment representation. This allows the model to properly distinguish between the query and the context.

The target of the encoder-only model consists of positions for the start and end of the span containing the attribute value. To prepare these for the training samples, we re-use the approach from the preprocessing procedure: we search for the target value (normalised and turned into a regular expression) in the segment representation. This gives us the exact index of the first and last character of the attribute value. Then, we tokenise the input sequences, and we use the `char_to_token` method of the `BertTokenizerFast` encodings to translate these character indices to token indices. If the attribute value is not present in the segment, we train the model to predict an empty span by assigning the `[CLS]` token as both the start and end of the span (i.e. $pos_{start} = pos_{end} = 0$). The loss function used to train the model is the average cross entropy loss over both the start and end positions.

The `BertForQuestionAnswering` model contains two output heads, one each for the logits of the start and end positions. To turn this into a probability

[4] `https://huggingface.co/SpanBERT/spanbert-base-cased`

distribution over all possible spans, we construct a matrix Σ, where $\Sigma_{i,j}$ contains our approximation of $P(pos_{start} = i, pos_{end} = j)$. To fill Σ, we apply the following procedure:

1. We mask all token positions that are not part of the context or the `[CLS]` token, to ensure the predicted span is always either the null response or part of the segment representation.
2. We compute the softmax over the start and end positions, to turn both of them into a tensor of probabilities. We obtain vectors σ_{start} and σ_{end}, where $\sigma_{start,i} \approx P(pos_{start} = i)$ and $\sigma_{end,j} \approx P(pos_{end} = j)$. For all masked positions p, we have $\sigma_{start,p} = \sigma_{end,p} \approx 0$.
3. We turn the two tensors into the $N \times N$ matrix $\Sigma = \sigma^T_{start} \times \sigma_{end}$. This initialises Σ such that $\Sigma_{i,j} = \sigma_{start,i} \cdot \sigma_{end,j}$, which is our estimate of $P(pos_{start} = i)P(pos_{end} = j)$ and, in turn, $P(pos_{start} = i, pos_{end} = j)$.
4. We set the probabilities $\Sigma_{i,j}$ to 0 for all $i > j$, as these cases indicate a situation where the end of the span is predicted to be somewhere before the start of the span.
5. We set the probabilities $\Sigma_{0,i}$ and $\Sigma_{i,0}$ to 0 for $i > 0$, to ensure the `[CLS]` token can only be included as part of the span if both the start and end positions point to it.

The remaining non-zero values represent the probabilities for all possible, valid spans. To turn this into a prediction, we select the combination of pos_{start} and pos_{end} that has the highest probability. The corresponding probability $\Sigma_{pos_{start},pos_{end}}$ is returned as the model's confidence.

Sequence-to-sequence

For the sequence-to-sequence (encoder-decoder) setup we used the Hugging Face `T5ForConditionalGeneration` model and `T5TokenizerFast` tokeniser. The `T5ForConditionalGeneration` model implements the standard T5 [25] model that is used for unified sequence-to-sequence learning. We initialise the model and tokeniser from the `t5-base` checkpoint.

Again, we encode the inputs to the model similar to the approach taken for T5 in the question-answering domain: we produce a single sequence where we use prefixes `attribute:` and `context:` to signify the different parts of the input. The same query we have seen above is now encoded as:

```
attribute: <attribute> context: <segment representation> </s>
```

Note that the T5 tokeniser uses `</s>` as the end of sequence token instead of `[EOS]`, and that the sequence does not contain any `[CLS]` or `[SEP]` tokens.

The target of the sequence-to-sequence model is the exact value of the attribute. Since the sequence-to-sequence model generates the output autoregressively, we do not need specific preprocessing to prepare the targets for training and we can simply use the tokenised targets to compute the loss and train the model. The loss is then the token-level cross-entropy loss. In the case that a segment does not contain the requested attribute, the model will output the empty sequence - more specifically, the sequence consisting of only the end of sequence token `</s>`.

To obtain the confidence for a prediction, we distinguish between two cases:

- The greedy decoding strategy averages over the probabilities of each predicted token in the generated sequence. This ensures the confidence of the model is invariant to the length of the prediction, and we only look at the likelihood that each token in the sequence is correct.
- The beam search strategy takes the final beam score of the generated sequence. This represents the confidence that the model has for this specific sequence compared to the other most likely candidates. As such, it is a good signal for the overall confidence of the model.

In our experiments, we implement a custom logits processor[5] to constrain the decoder output: we only allow it to generate token sequences that also occur in the tokenised context (as also mentioned in Section 4.1.3). As a result, the T5 model also extracts a span of the context, rather than purely generating it.

We have experimented with the usage of beam search (with a beam size of 3) in the output generation of the decoder. However, the gained performance increase was only minimal, and the beam search significantly increased the cost and duration of validation and evaluation steps. As a result, running the T5 experiments with beam search enabled became largely infeasible, and we refrained from using beam search in our final experiments.

[5] `https://huggingface.co/docs/transformers/internal/generation_utils#transformers.LogitsProcessor`

Chapter 5

Experiments and evaluation

The research questions introduced in Section 3.4 cannot be answered using theoretical or analytical approaches. Rather, to empirically measure the performance of our models in different configurations, we define and perform a variety of experiments. In this chapter, we describe each of these experiments and discuss the results obtained for each of them.

5.1 Experimental setup

For our experiments, we created a random train/validation/test split of the SWDE dataset with an 8:1:1 ratio. We then preprocessed the data with the preprocessing procedure explained in Section 4.2.1.

For every experiment, we train a separate model for each of the verticals in the dataset. Each model is trained for 50,000 gradient steps, where every training step is executed with an effective batch size of 64. To ensure an equal batch size throughout all our experiments, we apply gradient accumulation in the cases where we cannot fit the full batch size in our GPU's memory. In other words, if the batches that fit on the GPU are smaller than 64 samples, we accumulate the gradients of multiple of these batches (up until we've processed 64 samples) before we update the network's parameters. We use a linear learning rate scheduler with a base learning rate of $5 \cdot 10^{-5}$ and 1,000 warmup steps.

Every 1,000 training steps, we perform a validation step. Since performing validation on the entire validation set each time would greatly slow down training, we sample a random subset of 150 documents for every validation step. The validation metric we aim to optimise during training is the instance-level WM score. We use model checkpointing to save the best-performing model weights for each training run. After 20 validation steps

(i.e. 20,000 training steps) without an improvement in the WM score, we apply early stopping and terminate the training run.

After training has been completed, we reload the model checkpoint with the highest validation performance. We evaluate the model on the full validation and test set, as well as an equally sized random subset of the training set.

Since every page only contains a handful of mentions of the correct attribute value, most of the segments extracted from a page will not contain a meaningful value. To account for this imbalance during training, we apply a custom sampling strategy. We ensure half of the samples in a batch do contain a value for the requested attribute, and the other half does not. We also ensure that the attribute types for the segments that do contain a value are balanced: every attribute type occurs roughly as many times in a batch as all other attribute types.

Our experiments are conducted on NVIDIA GeForce RTX 3090 GPUs. A single full training run takes anywhere from 4 hours up to a full day, depending on the model architecture used and context size.

Similar to other recent research [37, 8, 40], we have not performed hyperparameter tuning for values like the learning rate, batch size or optimiser. We feel these hyperparameters can always be optimised as an additional improvement later on, so we decided to focus our research on experimentation with different representations and setups instead. We argue that these are likely to be more impactful than premature hyperparameter tweaking. Aside from that, each training setup requires 8 training runs of at least 4 hours each, meaning a hyperparameter sweep would become costly very quickly.

5.2 Experiment tracking

Because we run a large number of experiments, we invested into the machine learning operations (or MLops [32]) aspects of this part of the research. To schedule, monitor and archive each of our experiments, we make extensive use of Weights & Biases (W&B) [4]. We store all our datasets (i.e. the original dataset, the data splits and the preprocessed data) as Artifacts[1] in our W&B workspace. We log the training loss and all validation metrics to our workspace as well. After the final evaluation of a model, we update the run summary to contain the evaluation metrics for each split, and we store the segment- and document-level predictions as W&B tables.

Each experiment is set up as a W&B Sweep[2], which can be defined as a way to perform automatic hyperparameter searches. To start a sweep, we define

[1] `https://wandb.ai/site/artifacts`
[2] `https://wandb.ai/site/sweeps`

the necessary configuration values in a YAML file and initialise the sweep. Then, we can start 'agents' that will request runs from the server and execute them. After completing a training run, the agent requests a new run, all the way until the server indicates that all experiments have been completed.

The W&B sweeps are particularly useful to set up our experiments such that they are executed for each vertical. By including the vertical (and possibly other hyperparameters such as the architecture or context size) as a sweep parameter, we can start any number of agents and they will continue running until all experiments have been completed. This allows for optimal usage of resources, as our GPUs will not be sitting idle as long in between training runs.

Since all of our experiment tracking happens in a single W&B workspace, it is very easy to keep an overview of:

- The experiments we run, and how they are configured;
- The performance of each model;
- The predictions of each model;
- The training and validation history of each model;
- The dataset types and versions used for each experiment.

Collecting all experiments and data in a single workspace and publishing this workspace improves the reproducibility of our research, as an additional benefit.

5.3 Context sizes

As mentioned in Section 4.1.1, the context size of the segments likely affects the model's ability to extract different types of attributes. To investigate the differences in performance between context sizes and answer RQ II, we ran an experiment where we tested out different context sizes with the same model and representation. In these experiments, we used the encoder-only BERT model and the HTML representation introduced in Section 4.1.2. We varied the context sizes between 128, 256 and 512.

The results of this experiment can be found in Table 5.1. As can be seen, the instance-level performance increases slightly for larger context sizes (though at a cost of lowered precision), likely due to the model being able to use more of the surrounding information in its prediction. Table A.2 additionally displays the attribute-level performance of the models for each context size. Interestingly, the context size heavily influences the model's performance for specific attributes. For instance, the model with the lowest context size of 128 tokens outperforms the larger contexts for movie genre and restaurant

Table 5.1: Performance of a BERT model for different context sizes. Best performing models are highlighted in bold. The 'ensemble' model takes the predictions of each of the different context sizes and produces the one with the highest confidence.
Results marked with † are significantly ($p < 0.01$) higher than the model using a context size of 128.

	Global			Instance	
Context size	P	R	F_1	EM	WM
128	**0.96**	0.94	0.95	0.97	0.97
256	0.94	0.94	0.94	0.97	0.97†
512	0.94	0.94	0.94	0.97†	0.98†
Ensemble	0.95	**0.95**†	**0.95**†	**0.99**†	**0.99**†

address. The model with the largest context size of 512 tokens, on the other hand, clearly outperforms the others on movies' MPAA rating and NBA players' height and weight. Inspecting the differences in more detail, we notice that for some websites the HTML leaf nodes containing these fields are too large to fit in a context of 128 or 256 tokens. This highlights an issue with our segmentation approach, where important leaf nodes might be skipped if the HTML encoding becomes larger than the maximum context size.

The results for this experiment suggest that certain attributes benefit from a model that is focused on a smaller, local context (for a higher precision), while others need more of the surrounding context to allow for a good performance. The model that uses a context size of 256 tokens finds the middle ground between the two extremes and seems to provide a good balance between the performances of the smaller and larger context sizes.

Also, the quadratic memory requirement of the attention mechanism in our transformer-based models implies that doubling the context size also quadruples the amount of required memory. Since our GPUs can only fit a limited amount of data at a time, this forces us to reduce the batch size by a factor of four, which significantly slows down training. Because the performance gain for the largest context size of 512 is negligible, choosing for a context size of 256 ensures we can keep our training time reasonable. Hence, all subsequent experiments are performed using a context size of 256 tokens.

```
<div class="main container">
    <h1 id="title__pageTitle">This is a heading</h1>
    <p id="description">
    This is the description.
    </p>
</div>
```

Listing 5.1: The example HTML fragment from Listing 4.1, revisited.

5.4 Sequence representation

In Section 4.1.2, we discussed the different features of our segment representations. To address RQ III and test the usefulness of different types of representation, we train our models on different versions of the SWDE dataset, each preprocessed using a different representation. We distinguish between the following representations:

Text baseline To see the benefits of incorporating HTML information, we test a baseline system that contains only the textual contents in a segment. The HTML fragment in Listing 5.1 will be encoded as:

`This is a heading This is the description.`

HTML baseline This representation implements the 'simple' HTML encoding as discussed in Section 4.1.2 and implemented in [37]. It surrounds an HTML node's contents with an opening and closing tag and inserts the node's `id` and `class` attributes right after the opening tag. The resulting representation of Listing 5.1 is the following:

```
<div> class=main class=container <h1> id=title__pageTitle
↪ This is a heading </h1> <p> id=description This is the
↪ description. </p> </div>
```

Expanded HTML This representation is the main representation introduced in Section 4.1.2, where we encode the a most recent ancestors and their attributes. However, we experiment with various settings for this representation, to see the added benefit of each part of the representation. Specifically, we perform experiments to answer the following questions:

- Should we include the `id` and/or `class` of a node in the encoding of a node, alongside its tag?
- Should we encode the `id` and/or `class` for all types of nodes, or only for a subset of very general HTML tags that do not carry any semantic meaning? The intuition here is that tags like `h1` or `table` reveal more information about the contents of the node than generic tags such as `div` or `span`. By limiting the number of

tags that get a full encoding, we might fit a larger context window into a single segment, which, in turn, might improve the model's performance.

- Should we split attribute values into separate sub-components? For instance, the `id` of the `h1` tag in Listing 5.1 is a longer string that actually consists of three main parts: `title`, `page` and `title`. Tokenisation gives no guarantees about how these composed attribute values will be tokenised, so splitting them manually might improve the model's ability to recognise the different parts of the attribute value.

The results of these experiments are summarised in Table 5.2. Interestingly, the text baseline itself already performs well, which shows that the text surrounding an attribute value is quite indicative of its location in the document. The simple HTML baseline degrades performance, while our HTML encoding seems to help the models to perform better. Specifically, the inclusion of the `id` attribute increases performance, especially if we split its value into multiple sub-components. The inclusion of the `class` attribute improves the precision of our models, but seems to hurt the recall. After inspecting this drop in recall in more detail, we conclude that the inclusion of `class` attributes tends to cause the encoded sequence to exceed the maximum allowed length of 256 tokens. As a result, certain subtrees containing the target attribute value could no longer be encoded in the provided token limit, causing the model to miss the corresponding values altogether. Finally, we note that separating HTML attribute values into their sub-components marginally increases performance, while only encoding these values for a subset of tags hurts performance to a point below even the text baseline.

The decreased performance after including `class` attributes again highlights the problem with our approach that the length of the encoded sequence is bounded by the full attention mechanism in the transformer architecture. If we incorporate more information into the representation of a single segment, it becomes more likely that we can no longer fit certain subtrees into these sequences. For the model to perform optimally, we need to strike a balance between incorporating larger contexts and more encoded HTML information, while still keeping the sequence lengths within the desired limits.

Table 5.2: Performance of our information extraction models for the Movie domain, using different representations. Best results are marked in bold.
For our encoding, the 'ID' column specifies whether the `id` attribute was encoded, the 'Class' column specifies whether the `class` attribute was encoded, the 'Split' column specifies whether attribute values were separated into their sub-components, and the 'Subset' column specifies whether we limit the `id` and `class` encodings to `div`, `span`, `a` and `p` tags. For all configurations of our encoding, $a = 3$ ancestors are encoded.
Results marked with † are significantly ($p < 0.01$) higher than the model using our custom HTML encoding without including class or ID attributes. Results marked with ‡ are significantly ($p < 0.01$) higher than the text baseline.

	ID	Class	Split	Subset	Global			Instance	
					P	R	F_1	EM	WM
Text baseline					0.95	0.94	0.94	0.94	0.95†
HTML baseline					0.94	0.93	0.93	0.93	0.94
Our HTML encoding	-	-	-	-	0.95	0.94	0.94	0.94	0.95
	-	✓	-	-	0.96†‡	0.93	0.95	0.93	0.94
	-	✓	✓	-	0.96†‡	0.93	0.94	0.93	0.93
	✓	-	-	-	0.96†‡	0.95†‡	0.96†‡	0.95†‡	0.96†
	✓	✓	-	-	0.96†‡	0.93	0.94	0.93	0.93
	✓	✓	✓	-	0.96†‡	0.93	0.94	0.93	0.93
	✓	✓	✓	✓	0.94	0.92	0.93	0.92	0.93
	✓	✓	-	✓	0.94	0.92	0.93	0.92	0.93
	✓	-	✓	-	**0.96**†‡	**0.96**†‡	**0.96**†‡	**0.95**†‡	**0.96**†‡

5.5 Comparison with baselines

From the results in Sections 5.3 and 5.4, we conclude that the best configuration of our models is the setup in which we use a context size of 256 tokens and encode the HTML alongside the `id` value of each node split into sub-components. To answer RQ I, we need to evaluate how well our model with this setup performs compared to state-of-the-art information extractions models. Thus, we compare our model with the following recently proposed information extraction models, each of which has also been evaluated on the SWDE dataset.

WebFormer [34] A transformer architecture for ClosedIE that includes HTML nodes in the transformer blocks using graph attention. For text tokens, the model limits the memory requirement of the attention mechanism by employing relative attention, allowing for the efficient processing of larger documents. The model also includes attention patterns between HTML nodes and text nodes, learning a general understanding of the document. The model predicts the start and end of a span, similar to our encoder-only models.

WebKE [37] An OpenIE method that uses a pre-trained BERT model enriched with (1) extended positional embeddings, (2) 2D position information of HTML nodes in the document, and (3) inclusion of HTML start and end tags in the sequence. The model first uses the pre-trained encoder to extract areas of interest small enough to fully fit into a single sequence, after which it sequentially predicts OpenIE relations and their corresponding objects. Though the method is intended to be used for OpenIE, the paper also reports very high performance measurements for ClosedIE on the regular SWDE dataset.

SEP [6] A method that defines a set of context-free grammars to describe webpages and their contents. The paper shows that extraction is possible by using these grammars as wrappers and defines the task of finding a grammar that best describes a set of web pages. The grammars are generated by modelling the task as a search problem and applying the A^* search algorithm.

The results of our comparison can be found in Table 5.3. For WebFormer and SEP, we report the performances as reported in their respective papers. We were unable to reproduce WebFormer's results in spite of extensive efforts, including consultation of the original authors. We encountered SEP at a later time during this project and were unable to find a published method for reproducing their results. For WebKE, we could eventually reproduce their results, so we report both the performance from their paper and the results obtained by our own reproduction. This reproduction of WebKE is explained in detail in Section 5.6.

Table 5.3: Performance of our extraction models, compared to several strong baselines. Best results are marked in bold. For WebFormer and SEP, the reported numbers are taken from the respective papers. For WebKE, we report both the numbers from their paper and our reproduction of their numbers.

	Global			Instance	
Model	P	R	F_1	EM	WM
WebFormer	-	-	-	0.87	0.92
SEP	0.94	0.93	0.94	-	-
WebKE (theirs)	-	-	**0.97**	-	-
WebKE (ours)	0.79	0.65	0.67	0.65	0.67
Ours (BERT)	**0.95**	**0.95**	0.95	**0.98**	**0.98**
Ours (T5)	0.42	0.43	0.43	0.43	0.46

Note that WebFormer and WebKE apply a fully random train/validation/test split with an 8:1:1 ratio, similar to our approach. However, SEP generates a wrapper from a sample of only 20 pages per website, meaning it needs significantly less training data.

Immediately, we notice that the BERT models obtain high performance, while the T5 models stay behind. We are unsure why the sequence-to-sequence approach does not seem to work well, since it has been proven effective in the question-answering domain [25]. We hypothesise that the pre-training of T5 has taught it to produce sequences of natural language, while many of the attributes in the dataset do not follow the same patterns as natural language. As a result, the decoder is not quite capable of generating the correct sequences. For the remainder of this thesis, we therefore focus on the encoder-only BERT models.

We can see that our models can outperform the strong baselines of WebFormer and SEP. Our segmentation approach combined with full attention outperforms WebFormer's full document input with relative attention mechanisms - even with the added attention mechanisms representing the HTML structure. In fact, the text-only baseline (Table 5.2) also outperforms the results from WebFormer, indicating that the difference in performance lies not only in the method of incorporating HTML structure.

Comparison with SEP is less obvious, as their approach is substantially different from ours, and their models require significantly less training data. Nevertheless, our models seem to outperform the SEP baseline, albeit by a smaller margin than they outperform WebFormer.

WebKE as published does seem to outperform our models, but we were unable to reproduce their results ourselves. In the end, we were unable to find a conclusive explanation to these differences in performance. The comparison with WebKE is discussed in more detail in the next section.

5.6 WebKE reproduction

Since the authors of WebKE made their code publicly available, we were able to run their models and compare their results to ours in more detail. To do so, we downloaded their code from GitHub[3] and followed their instructions to download the preprocessed dataset and model checkpoints.

5.6.1 Data split

For a fair comparison, we applied the exact same data split that was used in their preprocessed dataset. The supplied preprocessed data consists of 24,879 files in the training set and 2,763 files in the test set. By taking the ground truth labels contained in the preprocessed data and comparing those with the labels from the Expanded SWDE dataset, we were able to uniquely map the vast majority of these files back to their corresponding webpages. For 22 pages in the training set ($\approx 0.09\%$) and 5 pages in the test set ($\approx 0.18\%$), we were unable to perform the reverse mapping to a unique page, as the provided labels overlapped with multiple pages. Since these ambiguous cases only made up a small portion of the dataset, we opted to discard these from our split altogether.

Similar to the approach taken for WebKE, we split the training set further into a training and validation set, so that we end up with the desired 8:1:1 split.

5.6.2 OpenIE to ClosedIE mapping

WebKE is an OpenIE system and, as such, was trained and evaluated on the Expanded SWDE dataset, which contains only the Movie, NBA player and University domains. As a result, WebKE does not output a single prediction for every type of attribute in the schema, but rather it outputs a $\langle r, o \rangle$ tuple for every relation and object on the page. To compare this system to our ClosedIE method, we need to map the OpenIE relations to the ClosedIE attributes we have been using, and vice versa. For each website in each vertical, we checked all the relations present in the Expanded SWDE dataset and mapped them back to their ClosedIE counterparts. The resulting mappings can be found in Appendix A.4.

[3] `https://github.com/redreamality/webke`

Table 5.4: Comparison between our model and WebKE on the regular SWDE dataset. Best results per vertical are marked in bold.
Results where our model performs significantly ($p < 0.01$) better than WebKE are marked with †. Results where WebKE performs significantly ($p < 0.01$) better than our model are marked with ‡.

		Global			Instance	
Vertical	Model	P	R	F_1	EM	WM
movie	WebKE	0.88	0.60	0.69	0.58	0.61
	Ours	**0.93**†	**0.88**†	**0.90**†	**0.87**†	**0.88**†
nbaplayer	WebKE	**0.92**‡	**0.61**‡	**0.74**‡	**0.61**	0.65
	Ours	0.56	0.56	0.56	0.59	**0.82**†
university	WebKE	0.94	0.75	0.81	0.76	0.76
	Ours	**0.96**†	**0.96**†	**0.96**†	**0.96**†	**0.98**†

For each of our experiments, we only evaluate WebKE and our models on the subset of attributes that exist in both the OpenIE and ClosedIE domains (per website).

5.6.3 Results

Since WebKE and our models have been trained on a different set of labels (of the Expanded SWDE and normal SWDE dataset, respectively), we compare the models on both datasets.

Comparison on SWDE

For the comparison on the regular SWDE dataset, we can use our own model's output as-is. For WebKE, we map the OpenIE labels in their predictions back to their ClosedIE counterparts and ignore all labels that do not exist in the OpenIE domain. To obtain a single prediction for each attribute (as opposed to the multiple labels that can be produced in OpenIE), we assume the most desirable situation for WebKE and choose the value that is most similar to one of the ground truth values.

With this setup, we can simply compute the global and instance-level metrics introduced in Section 2.2.2. As mentioned previously, we do not include attributes that do not exist in the OpenIE domain for a specific website. The comparison between our model and WebKE on the regular SWDE dataset can be found in Table 5.4.

As can be seen, our model significantly outperforms WebKE on the Movie and University domains. For the NBA player domain, our model obtains a higher instance-level WM score, though it performs poorly for the global metrics. While these results seem to imply that our setup is more effective than WebKE for two domains and less effective in the other, it is important to keep in mind that WebKE was trained as an OpenIE system on the Expanded SWDE dataset. We cannot draw a firm conclusion whether the differences in performance can be attributed to the different architectures, or whether they simply stem from the fact that both systems were trained with a different purpose and dataset.

Comparison on Expanded SWDE

To carry out the same comparison on the Expanded SWDE dataset, we map the OpenIE labels that occur in the WebKE predictions, as well as the Expanded SWDE ground truth labels, to their ClosedIE variants, again ignoring any relations that do not exist in the OpenIE domain. We convert our predictions to the correct format by taking each available attribute per document and, if we predict a non-empty value for that attribute, converting it to a $\langle r, o \rangle$ tuple. As we did earlier, we skip attributes for websites that do not have an OpenIE counterpart.

Given these sets of tuples per document, we normalise the predicted and ground truth attribute values and compute the precision, recall and F_1 over all tuples, macro-averaged over all documents. The normalisation step was not performed in the original evaluation of WebKE. However, due to the differences in training data, our model outputs values for an attribute in a different format than WebKE, such that a fully exact match is rare and hard to achieve. We discuss these discrepancies in more detail in Section 5.6.4.

The results of our comparison on the Expanded SWDE dataset can be found in Table 5.5. In this case, our model outperforms WebKE for the Movie domain, while WebKE performs better in the NBA player and University domains. Note that, in contrast to the evaluation on the regular SWDE dataset, WebKE has an inherent advantage in this evaluation - both due to its ability to output multiple values per attribute and because it was trained on the OpenIE task's data.

5.6.4 Discussion

As mentioned for both evaluation settings, the one-to-one comparison between our model and WebKE is not straightforward. The fact that both models were trained on different datasets with a different set of labels implies it is unfair to evaluate them on the ClosedIE setting only, and we have to include the Expanded SWDE dataset for our evaluation.

Table 5.5: Comparison between our model and WebKE on the Expanded SWDE dataset. Best results per vertical are marked in bold.
Results where our model performs significantly ($p < 0.01$) better than WebKE are marked with †. Results where WebKE performs significantly ($p < 0.01$) better than our model are marked with ‡.

Vertical	Model	P	R	F_1
`movie`	WebKE	**0.80**‡	0.48	0.60
	Ours	0.79	**0.63**†	**0.70**†
`nbaplayer`	WebKE	**0.69**‡	**0.66**‡	**0.68**‡
	Ours	0.66	0.59	0.62
`university`	WebKE	**0.90**‡	**0.95**	**0.92**‡
	Ours	0.68	0.94	0.79

We compared the labels of both the regular and Expanded SWDE datasets and noticed that both datasets suffer from representation problems. In the regular SWDE dataset, values can contain HTML entities (such as ` `) or unnecessary prefixes (e.g. '*Weight:*'), while the labels in the Expanded SWDE dataset often contain meaningless and superfluous whitespace.

For the two versions of the SWDE dataset specifically, it turns out that values for the same attribute and the same website can differ, unexpectedly. For instance, the regular SWDE dataset can include a prefix '*Height:*' in the attribute value, while the Expanded SWDE dataset uses it as the predicate. As a result, our model is trained to include the prefix in the output, while WebKE is not. Consequently, two models trained on the same websites with similar goals are producing different outputs, and depending on the dataset used, either could be correct.

We argue that the inclusion of these HTML artefacts in the dataset distracts from the goal of the models: extracting values for specific attributes from a document. Normalising the attribute values in the metrics (e.g. removing whitespace and punctuation) may alleviate this problem for evaluation purposes to some degree, but does not take away the fact that models are trained to also predict these HTML artefacts. Possible solutions could be to clean the labels from the existing datasets or to create new datasets entirely. With the current abundance of structured data annotation on the Web, it should be feasible to construct new, standardised datasets for structured data extraction with minimal effort (as was also done in [34]).

Nevertheless, if we compare our own system to WebKE analytically, we encounter several major differences. WebKE's preprocessing requires visual

rendering of each document in a web browser to compute positional information of all HTML elements. As discussed in Section 4.1.4, this approach is costly and not robust to changes in the configuration of the chosen web browser. Also, the extended positional information and vocabulary used by WebKE require access to a non-standard BERT model and encoder, while our own approach simply fine-tunes a standard pre-trained BERT model. Finally, WebKE is pre-trained on the webpages in the SWDE dataset, which might give it a hidden advantage in evaluation on both the regular and Expanded SWDE datasets.

Although the empirical comparison between our model and WebKE is inconclusive due to differences between the regular and Expanded SWDE datasets, our model seems to be easier to set up and use - both because preprocessing is less costly and because it only requires fine-tuning of a standard pre-trained language model. This simplicity makes it more attractive to be used in a practical setting.

5.7 Zero-shot setting

In the ideal situation, we would train our models on a set of web pages with structured data annotations and use them on new, unseen websites. To test the generalisation capabilities of our models (RQ IV), we therefore continue to perform experiments in a zero-shot setting. For each vertical, we split the websites into 5 folds of 2 websites each. We perform a cross-fold validation where we use one fold for testing, one for validation, and the remaining 3 folds (6 websites) are used for training. We then average our performance over 4 different setups. To evaluate our zero-shot performance, we compare our model against the following baselines:

LANTERN [40] A model that learns representations for each HTML node by incorporating textual embeddings, discrete page features and inter-node relationships. Extraction is defined as a node tagging task, where classification is performed for each node to determine the attribute it contains (if any).

DOM-LM [8] An encoder-only transformer model that inserts several features of the HTML structure into the encoder model by adding them to the token and positional embeddings. To split the document into subtrees that are small enough to be handled by the transformer, a sliding window is passed over the tree structure, adding and pruning certain nodes to stay within a certain token limit. Extraction is again defined as a node tagging task, where separator tokens specify the start of each node's content and classification is performed for each of these separator tokens.

Table 5.6: Performance of our extraction models in the zero-shot setting, compared to two strong baselines. Best performances are marked in bold. For the baseline models, we report the numbers from their respective papers.

	Global			Instance	
	P	R	F_1	EM	WM
LANTERN	-	-	**0.94**	-	-
DOM-LM	-	-	0.78	-	**0.96**
Ours	**0.66**	**0.60**	0.62	**0.60**	0.69

Unfortunately, both LANTERN and DOM-LM only report performance for few-shot and zero-shot settings, which is why we could not include them in the regular evaluation setting. WebFormer, SEP and WebKE, on the other hand, do not report performance for the zero-shot setting, meaning we cannot include them in our comparison of zero-shot performance. We were also unable to produce zero-shot results for WebKE ourselves, as their pre-trained models were already trained on all websites in the dataset.

The results of the zero-shot experiment are summarised in Table 5.6. For both baselines, we report the results that were reported in their respective papers. For LANTERN, we use the performance of the models that were trained on 5 seed websites and evaluated on the remaining 5 websites. For DOM-LM, we report the performance of the models that were trained on 10% of the data of 5 seed websites, validated on a different website and evaluated on the remaining 4 websites. Our models were trained on data from 6 websites, and 2 websites were used for validation and testing each.

From these results, we can see that our models do not generalise as well as both baselines. We hypothesise that, during training, our models have learned the specific structure of documents in the training set, rather than more general interpretations of the HTML structure and content. This explains why our models do perform well in the regular case but fall behind in the zero-shot setting. Part of this could also be explained by the limited context size and HTML information we can incorporate in our limited sequence length, preventing our models from learning general, long-distance dependencies in the DOM tree.

In Appendix A.3, we provide the zero-shot performance of our models per attribute. We see that performance is high for attributes like website and ISBN. This is likely because these values have very specific formats, and as such are more easily recognisable from their value alone. More generic values like movie titles suffer most in the zero-shot setting, again highlighting our model's difficulty with understanding more general structural elements.

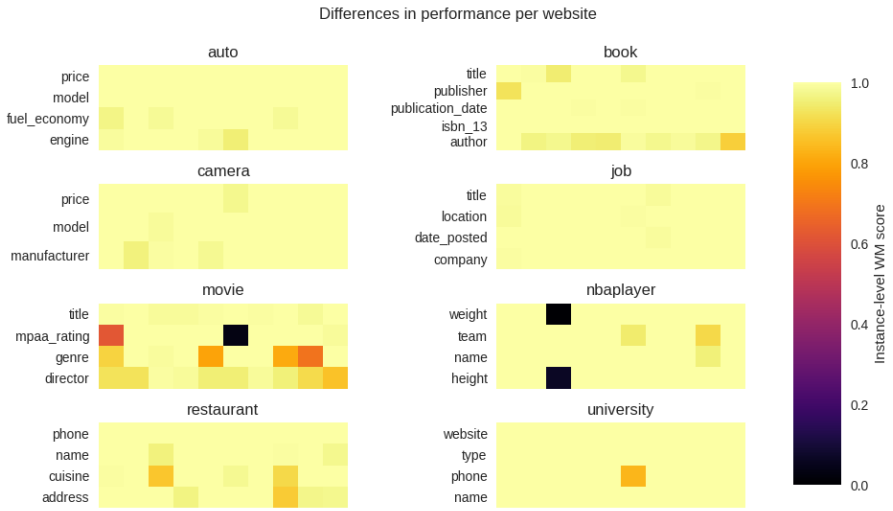


Figure 5.1: Instance-level WM performance for each website per vertical and attribute.

5.8 Failure analysis

In this section, we perform a more in-depth failure analysis of our models. Specifically, the goal is to figure out in what cases our models fail, and why.

5.8.1 Performance per website

While the metrics introduced in Section 2.2.2 provide us with a useful method of comparing different approaches, they inherently summarise the performance of the models over all verticals, attributes and websites. They give us the information that our method fails in roughly 2-5% of the cases, yet they do not indicate whether these failures are evenly distributed over all verticals and websites, or whether they occur mostly in specific cases.

To gain more insight into this distribution, we plot the performance of our models for each website, per vertical and attribute in Figure 5.1. We can clearly see that the failures are not evenly distributed: failures are mostly found in specific websites. Similarly, the attribute-level performance (Table A.1) shows optimal performance for many attributes but poor performance for some others (such as NBA player height or weight). This indicates that failures do not simply occur randomly. Rather, there are structural issues with certain websites and attributes that our models have trouble dealing with. In the next section, we investigate these issues in more depth.

5.8.2 Failure types

To gain a better understanding of the types of encountered failures, we inspect a random sample of failures and categorise them into different types. For each attribute per vertical, we sample 20 failures, where we denote a prediction as a failure if the exact match (instance-level EM) metric is equal to 0. The complete overview of sampled failures can be found in Table A.7.

We distinguish the following failure types:

Incorrect value The model has produced a sensible value, but it is incorrect for the requested attribute. An example of this type of failure is the case where the model extracts an author name from the webpage, but they are not the author of the book that is the subject entity of the page. Likewise, the model can extract a correct cuisine from a document, but it does not fit the restaurant that the page is about.

Incomplete value The model was able to find the correct value on the page, but it only extracted part of it.

Missed attribute The model could not find a value for the requested attribute, even though it is mentioned on the page.

Nonexistent attribute The model has produced a value for the requested attribute, but the page does not contain a ground truth value for that attribute.

Nonsense The model has produced a value for the attribute, but it does not represent a sensible or possibly correct value for it.

HTML encodings The value extracted by the model contains artefacts of our HTML encoding, such as `p1t li p0t a`.

Normalisation error The value extracted by our model is actually correct, but the normalisation step of the EM metric subtly fails, causing the EM metric to report it as an error instead. This happens, for instance, when different types of quotation marks are used (e.g. ’ instead of ') that are not discarded in the punctuation removal. Or, the ground truth value could contain unicode characters that are not present in the predicted values, but which are also not removed when normalising the values.

Multiple values The span extracted by our models actually contains multiple values, all of which are correct for the requested attribute.

Spelling mismatch The ground-truth and predicted values actually contain the same value, but they are spelled in a different way.

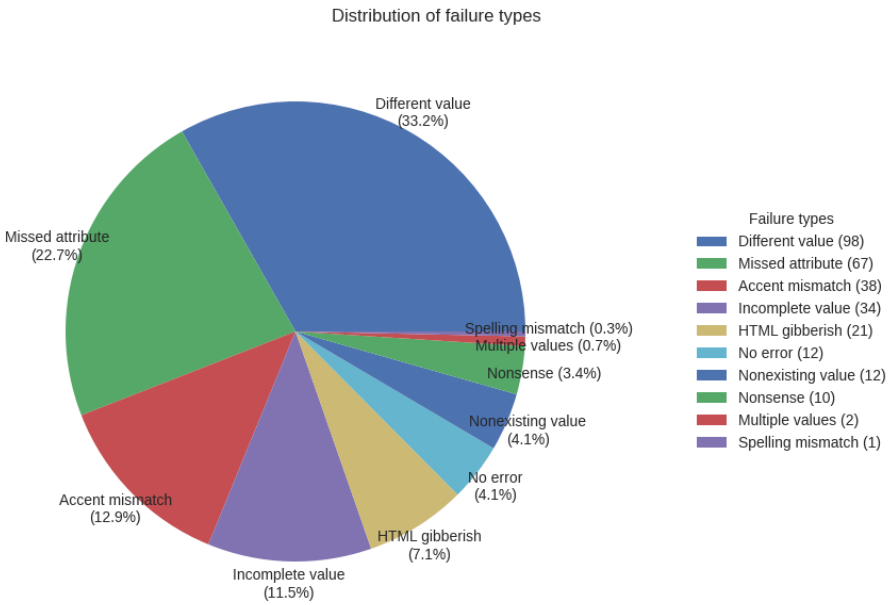


Figure 5.2: Distribution of failure types.

The distribution of each of the failure types is visualised in Figure 5.2. As can be seen, almost a third of the failures can be attributed to the model selecting values that appear sensible the requested attribute, but are incorrect for the current document. These are the cases in which the model appears to be focusing on the meaning of the value, rather than the surrounding HTML context. It could be that the correct value is found elsewhere on the page, but is lost in the reranking step. We analyse this situation in more detail in Section 5.8.3.

Another large portion ($\sim 23\%$) of the failures are caused by the model missing attribute values altogether. A large part of these missing attribute failures occur for specific websites and attributes. For instance, the movies' MPAA rating is often missed on IMDb, and the NBA players' height and weight are usually missed on the Fox Sports website. After inspecting these specific failures more closely, we conclude that these values are missed because the surrounding context is too large to be represented in a single HTML segment. By solving this inherent shortcoming of our approach (e.g. by moving to a different segmentation approach or adjusting the way in which HTML context is incorporated in the input), it should be possible to resolve these failures.

The normalisation errors, which account for roughly 17% of the failure cases, actually point to an error in the normalisation process part of computing the metrics. While reported as failures by our metrics (and not unlikely, also

reported as such by other papers using the SWDE dataset), they should in fact not be counted as erroneous predictions.

In the cases where our model produces incomplete attribute values ($\sim 12\%$), it is actually able to accurately find the location of the correct attribute value. However, the span-based extraction approach allows for cut-offs at incorrect positions. This can be resolved by switching from a span-based extraction method to a node tagging approach (similar to [40, 8]).

After inspecting the failures in which parts of our HTML encodings are present, we note that both the start and the end of the extracted span contain correct values for the requested attributes. Hence, these cases actually fall in the same category as the 'multiple value' instances, adding up to roughly 8% of the total failures. These errors can be explained by the way in which we predict the start and end positions of the spans. Because the start and end logits are computed separately, the model assumes their probabilities are independent. In practice, though, they are highly dependent. If an attribute has several correct possible values in a sequence, the start and end positions should both refer to the same attribute value, instead of independently choosing different values and resulting in a prediction spanning multiple values. To alleviate this problem, we could condition the end position probabilities on the start position, ensuring they are not computed independently. Alternatively, opting for a node tagging approach instead of span selection would remove this problem altogether.

The remaining 8% of failures seem to be semi-random quirks of our model. The nonexistent values appear to occur when a value is found that seems suitable for the requested attribute, rather than the model focusing on specific areas of the HTML document and realising the requested attribute does not appear on the page. Most of the nonsensical values are predictions of the location 'al', which might be caused by the model learning through a set of examples that that specific combination of letters is short for the state of Alabama. Likewise, the prediction of the numerical value 543 for a 'location' field could be explained as the model trying to predict a postal code, even though it is incorrect in this instance. Again, these issues highlight our model's tendency to focus on the semantic value of an attribute, rather than using specific cues in the HTML structure to find the correct value.

5.8.3 Reranking failures

As explained in Section 4.1.4, our model tries to extract a value from each segment of an HTML document and aggregates the resulting values to obtain a final prediction. As a result, it is possible that a correct value is extracted from a segment, but another (incorrect) value with a higher confidence is produced as output instead. This could explain some of the failures we

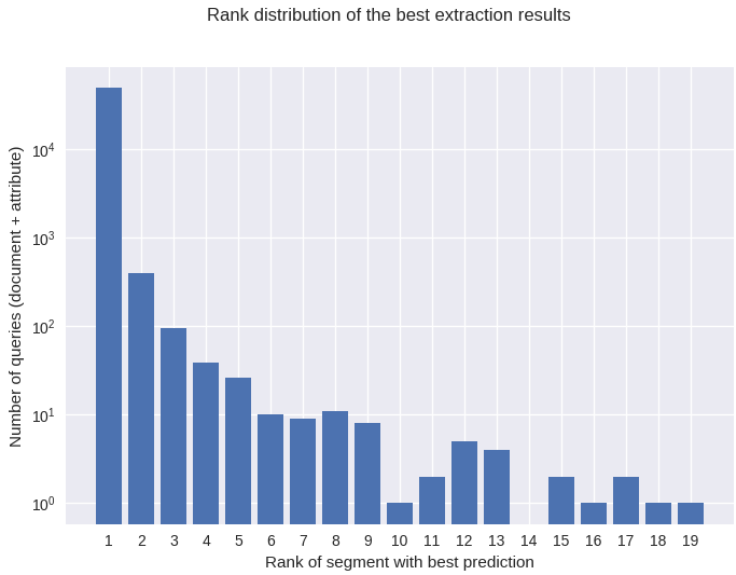


Figure 5.3: Rank distribution of the segments with a correct predicted value.

saw in the previous section. To gain some more insight into the number of times this happens, we compute the *rank* of the correct prediction for each document and attribute. In other words, we sort the segments and their predictions for each document by the model's confidence score, and then compute the 1-based index of the segment containing the best prediction (i.e. the one with the highest instance-level WM score).

The distribution of these ranks is shown in Figure 5.3. In an ideal ranking, where the confidence scores perfectly order the segments in terms of accuracy, all ranks should be equal to 1. After all, we only output the correct value if the corresponding segment has the highest confidence. However, we can clearly see that there are many cases where the rank is higher than 1, each of which will result in the model producing an incorrect prediction. Table A.8 additionally displays the attribute-level mean reciprocal rank (MRR) of the correct segment predictions, showing that reranking only fails for a handful of specific attributes such as book authors, movie directors or genres, and restaurant addresses. From Table A.7, we see that these attributes are the ones for which our models often produce sensible values that are incorrect for the entity at hand (that we denote with failure type 'incorrect value'). Since these values seem correct for the given attribute, it makes sense that the model assigns them a high confidence score. Ideally, though, the model should be able to use cues of the surrounding HTML structure to see that the value seems correct but is in a weird position on the page, in turn lowering its confidence for that specific value.

To improve the performance of the model in the cases where the confidence scores are not sufficient for a proper ranking, a more sophisticated reranker could also prove helpful (see Section 7.3).

5.9 Discussion

Through our experiments, we have encountered the following answers to each of our research questions.

RQ I. Our models perform well empirically on the regular extraction task. We have seen that the segmentation approach works well for extraction purposes - better than using the full-document input with a relative attention mechanism like WebFormer does.

a) With our setup, the encoder-only approach outperforms the sequence-to-sequence methods on the information extraction task by a large margin.

RQ II. The segment size must strike a balance between being long enough to allow for larger HTML subtrees and more HTML information to be encoded, but short enough to keep the model's memory footprint reasonable. Depending on the attribute and the web pages at hand, either a smaller or a larger context size could prove more effective.

RQ III. The inclusion of HTML information into the sequence improves our models beyond using textual content only. Our custom encoding outperforms the simple HTML baseline of including opening and closing HTML tags in-order, and we have shown empirically that the inclusion of `id` and `class` attributes (and splitting them into sub-components) increases the performance of the model. However, including too much of the HTML information can cause the representation of certain nodes to exceed the maximum allowed context size, and the models could miss attribute values in these nodes as a result.

RQ IV. At the moment, our models do not generalise well enough to new, unseen websites to be used in practice. Zero-shot performance is better for certain attributes with a specific format, but in general our models do not acquire enough understanding of the page structure that they can leverage when performing extraction on new websites.

The major shortcoming of our approach - which came up in our experiments multiple times - is that the limited sequence length poses a limitation on the amount of information that can be encoded in our sequences. As a result, we have to carefully decide which information to include in our representation lest we pass this length limit and miss out on certain attributes. Our current setup already suffers from these problems, missing certain attributes for certain websites because the relevant context cannot be encoded in a single segment. The limited context span (and thus, limited amount of HTML information that can be encoded) have as a result that our models have trouble generalising to new, unseen websites.

To counteract these issues, we might need to rethink how HTML information is represented in our models. An idea to significantly reduce the length of the encoded sequences is to inject the relevant HTML information into the embeddings passed to the encoder rather than inserting them in the sequence. This approach is also taken in [8], but they use a set of hand-crafted features limited to superficial information on the HTML structure. Ideally, we would also include the values of the `id` and `class` attributes of each node. In initial exploratory experiments, we used our HTML encodings with SBERT [27] to transform these encodings into embeddings. By adding these to the token and position embeddings, we were able to obtain promising results. However, due to time constraints, we were unable to explore this alternative representation in more detail, and are forced to leave it up for future work.

Chapter 6

Conclusions

From our results in Chapter 5, it is clear that our models perform on par with state-of-the-art models in recent literature - sometimes even surpassing them. However, the results in the zero-shot domain show that we still need improvements in that area before these models can be used in a practical setting. The ideal setup of training the models on websites containing structured microformat annotations (as described in Section 3.2) requires high performance on new, unseen websites, and our models do not generalise well enough to be useful for this purpose yet. Other approaches [8, 40] have shown promising results in that regard, and can be combined with our method.

Nevertheless, we have shown our approach of segmenting a document into its subtrees and processing each segment individually generally performs better than handling the full document at a time. Even in the case where we do not incorporate HTML structure, we can outperform several recent and much more complicated baselines. Our findings also show empirically that the inclusion of HTML structure in the input sequence is beneficial to the extraction capabilities of an extraction model, and increases performance beyond the text-only approach.

Our experiments with the SWDE (and Expanded SWDE) dataset have revealed that the ground truth values in both datasets are of a very specific format and include artefacts of the input HTML. Examples of these artefacts include encoded HTML entities, superfluous whitespace, and incomplete attribute values. As a result, models trained and evaluated on these datasets are implicitly also evaluated on their ability to copy these artefacts, which is not a desirable property for models in practical applications. It makes a comparison between results on the two different datasets difficult, even if the underlying data remains the same. We conclude that future research might benefit from evaluating on alternate datasets - alongside the SWDE dataset - similar to [10, 34, 8, 6].

Chapter 7

Future work

Although our models perform well empirically, throughout this thesis we have mentioned several limitations of our approach. In this chapter, we discuss these limitations in more detail and suggest possible ways to solve them in future research.

7.1 Evaluation on alternative datasets

All of our experiments were performed and reported on the SWDE dataset. In the literature, this dataset has been established as the default for information extraction tasks, which makes it highly suitable for comparison against other state-of-the-art approaches. However, the dataset was created over a decade ago, and the Web has changed significantly in recent years - both in terms of content and in terms of page structure. This decreases the meaningfulness of evaluating on the SWDE dataset, as its contents and labels might no longer be representative of the current state of websites containing structured information. Besides that, we have seen that the labels for the SWDE dataset contain several artefacts from the HTML structure that affect the evaluation, through aspects that should not matter in an information extraction system.

To account for these disadvantages of the SWDE dataset, it would be insightful to also evaluate our models and other baselines on alternative (newer) datasets. That way, we would test the model's performance on a more recent sample of the Web, which gives more insight into the model's validity, i.e. the viability of using the model in a practical application.

A handful of suitable datasets have been developed recently, such as the Klarna Product Page Dataset [15] and the PLAtE dataset [28]. An attractive alternative is to use Schema.org microformat annotations (e.g. from the

WebDataCommons project [22]) to generate a dataset containing a wide variety of verticals and entity types. This also fits in nicely with the setup proposed in Section 3.2, where we discuss the possibility of using these annotations as training data for real-world applications. An additional benefit is that the data from WebDataCommons is updated yearly, making it easier to generate datasets that accurately reflect a recent sample of the Web.

7.2 Multi-value attributes and multi-entity pages

Our current ClosedIE setup only allows the model to extract a single value per document, for any given attribute. However, this is not always a fair assumption, as many attributes can realistically have multiple values. For instance, a movie can have multiple directors or actors, and a system that can only extract a single value might provide an end user with incomplete information. In other situations, it is even more important to retrieve multiple values per attribute. For example, given a recipe, both the list of ingredients and the list of tasks need to be extracted completely, or the recipe will no longer make sense.

To achieve this, our architecture needs to be adjusted to output multiple values per attribute. This could be done by predicting multiple spans per segment and adjusting the aggregation method to output multiple values as well. To extract multiple spans from a given sequence, we could change the softmax in the start and end position logits to a sigmoid function to obtain probabilities per token, and apply a threshold to keep only those spans that the model is confident enough about.

Another approach would change the purpose of the model from a span extraction task to a node classification task, where each node is either labelled as containing an attribute value or not. With this approach, the aggregation method simply collects the text contents of each node that contains an attribute value.

In a similar vein, our current method only supports single-entity documents, such as webpages about a specific movie or book. To allow for a larger amount of information to be extracted from the Web, it would be useful if our model could handle multi-entity pages as well, such as pages that contain an overview of popular movies or a list of books for sale. The suggestions for extracting multi-value attributes should also make it possible to extract attributes for multiple entities in a single document, though the aggregation step should be adjusted to assign each attribute value to the correct subject entity.

7.3 Reranking segment predictions

As we have seen in Section 5.8.3, our models may extract the correct value of the attribute from a specific segment, but then encounter a higher confidence for an (incorrect) value in another segment. In that case, our prediction will be incorrect, even though the value could have been extracted successfully.

To remedy this problem, we could add an improved reranker in the aggregation step. Its purpose would be to take the segments, their predictions and possibly other useful information (such as the BERT embeddings of the segment or its position on the page) and rerank the extracted results based on the likelihood that each of them is correct. As a result, we should no longer have the issue that our model outputs an incorrect value if the correct value was found somewhere else on the page.

7.4 Generalisation across websites

In Section 5.7, we have shown that our model's performance in the zero-shot setting is inadequate. We hypothesised that this is due to our model learning very concrete, easily identifiable features of the HTML documents instead of learning more general representations. As a result, our models overfit on the specific HTML structure of the training set and lack the generalisability to perform well for unseen websites.

A possible solution to this problem could be to develop more general features for the HTML contents instead of the a most recent ancestors we are currently using. Alternatively, we could try to (pre-)train the models with data from a larger selection of websites, which should allow them to pick up on more general patterns and focus less on the layout of only a handful of websites. Similarly, we could experiment with possible data augmentation strategies to add more variations to the HTML structure of the training data, achieving the same goal but without the need for a large selection of different websites in the training set.

Bibliography

[1] Web Data Commons Extraction Report - October 2021 Corpus. URL: http://webdatacommons.org/structureddata/2021-12/stats/stats.html.

[2] Joshua Ainslie, Santiago Ontañón, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. ETC: Encoding Long and Structured Inputs in Transformers. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 268–284. Association for Computational Linguistics, 2020. doi:10.18653/v1/2020.emnlp-main.19.

[3] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The Long-Document Transformer. *CoRR*, abs/2004.05150, 2020. arXiv: 2004.05150. URL: https://arxiv.org/abs/2004.05150.

[4] Lukas Biewald. Experiment Tracking with Weights and Biases, 2020. URL: https://www.wandb.com/.

[5] Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Extraction and Integration of Partially Overlapping Web Sources. *Proc. VLDB Endow.*, 6(10):805–816, 2013. URL: http://www.vldb.org/pvldb/vol6/p805-bronzi.pdf, doi:10.14778/2536206.2536209.

[6] Valerio Cetorelli, Paolo Atzeni, Valter Crescenzi, and Franco Milicchio. The Smallest Extraction Problem. *Proc. VLDB Endow.*, 14(11):2445–2458, 2021. URL: http://www.vldb.org/pvldb/vol14/p2445-crescenzi.pdf, doi:10.14778/3476249.3476293.

[7] Andrew M. Dai and Quoc V. Le. Semi-supervised Sequence Learning. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages

3079–3087, 2015. URL: https://proceedings.neurips.cc/paper/2015/hash/7137debd45ae4d0ab9aa953017286b20-Abstract.html.

[8] Xiang Deng, Prashant Shiralkar, Colin Lockard, Binxuan Huang, and Huan Sun. DOM-LM: Learning Generalizable Representations for HTML Documents. *CoRR*, abs/2201.10608, 2022. arXiv: 2201.10608. URL: https://arxiv.org/abs/2201.10608.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019. doi:10.18653/v1/n19-1423.

[10] John Foley, Michael Bendersky, and Vanja Josifovski. Learning to Extract Local Events from the Web. In Ricardo Baeza-Yates, Mounia Lalmas, Alistair Moffat, and Berthier A. Ribeiro-Neto, editors, *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015*, pages 423–432. ACM, 2015. doi:10.1145/2766462.2767739.

[11] R. V. Guha, Dan Brickley, and Steve MacBeth. Schema.org: Evolution of Structured Data on the Web: Big data makes common schemas even more necessary. *Queue*, 13(9):10–37, November 2015. doi:10.1145/2857274.2857276.

[12] Pankaj Gulhane, Amit Madaan, Rupesh R. Mehta, Jeyashankher Ramamirtham, Rajeev Rastogi, Sandeepkumar Satpal, Srinivasan H. Sengamedu, Ashwin Tengli, and Charu Tiwari. Web-scale information extraction with vertex. In Serge Abiteboul, Klemens Böhm, Christoph Koch, and Kian-Lee Tan, editors, *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 1209–1220. IEEE Computer Society, 2011. doi:10.1109/ICDE.2011.5767842.

[13] Qiang Hao, Rui Cai, Yanwei Pang, and Lei Zhang. From one tree to a forest: a unified solution for structured web data extraction. In Wei-Ying Ma, Jian-Yun Nie, Ricardo Baeza-Yates, Tat-Seng Chua, and W. Bruce Croft, editors, *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2011, Beijing, China, July 25-29, 2011*, pages 775–784. ACM, 2011. doi:10.1145/2009916.2010020.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, 1997. doi:10.1162/neco.1997.9.8.1735.

[15] Alexandra Hotti, Riccardo Sven Risuleo, Stefan Magureanu, Aref Moradi, and Jens Lagergren. The Klarna Product Page Dataset: A Realistic Benchmark for Web Representation Learning. *CoRR*, abs/2111.02168, 2021. arXiv: 2111.02168. URL: https://arxiv.org/abs/2111.02168.

[16] Gautier Izacard and Edouard Grave. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering, February 2021. Number: arXiv:2007.01282 arXiv:2007.01282 [cs]. URL: http://arxiv.org/abs/2007.01282.

[17] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. SpanBERT: Improving Pre-training by Representing and Predicting Spans. *Trans. Assoc. Comput. Linguistics*, 8:64–77, 2020. URL: https://doi.org/10.1162/tacl_a_00300, doi:10.1162/tacl_a_00300.

[18] Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In Eduardo Blanco and Wei Lu, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*, pages 66–71. Association for Computational Linguistics, 2018. doi:10.18653/v1/d18-2012.

[19] Nicholas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper Induction for Information Extraction. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*, pages 729–737. Morgan Kaufmann, 1997.

[20] Colin Lockard, Xin Luna Dong, Arash Einolghozati, and Prashant Shiralkar. CERES: Distantly Supervised Relation Extraction from the Semi-Structured Web. *Proc. VLDB Endow.*, 11(10):1084–1096, June 2018. Publisher: VLDB Endowment. doi:10.14778/3231751.3231758.

[21] Colin Lockard, Prashant Shiralkar, and Xin Luna Dong. OpenCeres: When Open Information Extraction Meets the Semi-Structured Web. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1*

(Long and Short Papers), pages 3047–3056. Association for Computational Linguistics, 2019. doi:10.18653/v1/n19-1309.

[22] Hannes Mühleisen and Christian Bizer. Web Data Commons - Extracting Structured Data from Two Large Web Corpora. In Christian Bizer, Tom Heath, Tim Berners-Lee, and Michael Hausenblas, editors, *WWW2012 Workshop on Linked Data on the Web, Lyon, France, 16 April, 2012*, volume 937 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012. URL: http://ceur-ws.org/Vol-937/ldow2012-inv-paper-2.pdf.

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. page 12.

[24] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training.

[25] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020. URL: http://jmlr.org/papers/v21/20-074.html.

[26] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In Jian Su, Xavier Carreras, and Kevin Duh, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2383–2392. The Association for Computational Linguistics, 2016. doi:10.18653/v1/d16-1264.

[27] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3980–3990. Association for Computational Linguistics, 2019. doi:10.18653/v1/D19-1410.

[28] Aidan San, Jan Bakus, Colin Lockard, David M. Ciemiewicz, Yangfeng Ji, Sandeep Atluri, Kevin Small, and Heba Elfardy. PLAtE: A Large-

scale Dataset for List Page Web Extraction. *CoRR*, abs/2205.12386, 2022. arXiv: 2205.12386. doi:10.48550/arXiv.2205.12386.

[29] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 9895–9901. Association for Computational Linguistics, 2021. doi:10.18653/v1/2021.emnlp-main.779.

[30] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016. doi:10.18653/v1/p16-1162.

[31] Yusuke Shibata, Takuya Kida, Shuichi Fukamachi, Masayuki Takeda, Ayumi Shinohara, Takeshi Shinohara, and Setsuo Arikawa. Speeding up pattern matching by text compression. In *Italian Conference on Algorithms and Complexity*, pages 306–315. Springer, 2000.

[32] Mark Treveil, Nicolas Omont, Clément Stenac, Kenji Lefevre, Du Phan, Joachim Zentici, Adrien Lavoillotte, Makoto Miyazaki, and Lynn Heidmann. *Introducing MLOps*. O'Reilly Media, 2020.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[34] Qifan Wang, Yi Fang, Anirudh Ravula, Fuli Feng, Xiaojun Quan, and Dongfang Liu. WebFormer: The Web-page Transformer for Structure Information Extraction. In Frédérique Laforest, Raphaël Troncy, Elena Simperl, Deepak Agarwal, Aristides Gionis, Ivan Herman, and Lionel Médini, editors, *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, pages 3124–3133. ACM, 2022. doi:10.1145/3485447.3512032.

[35] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *CoRR*, abs/1910.03771, 2019. arXiv: 1910.03771. URL: `http://arxiv.org/abs/1910.03771`.

[36] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR*, abs/1609.08144, 2016. arXiv: 1609.08144. URL: `http://arxiv.org/abs/1609.08144`.

[37] Chenhao Xie, Wenhao Huang, Jiaqing Liang, Chengsong Huang, and Yanghua Xiao. WebKE: Knowledge Extraction from Semi-structured Web with Pre-trained Markup Language Model. In Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong, editors, *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 2211–2220. ACM, 2021. `doi:10.1145/3459637.3482491`.

[38] Song Xu, Haoran Li, Peng Yuan, Youzheng Wu, Xiaodong He, and Bowen Zhou. Self-Attention Guided Copy Mechanism for Abstractive Summarization. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 1355–1362. Association for Computational Linguistics, 2020. `doi:10.18653/v1/2020.acl-main.125`.

[39] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big Bird: Transformers for Longer Sequences. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL: `https://proceedings.neurips.cc/paper/2020/hash/c8512d142a2d849725f31a9a7a361ab9-Abstract.html`.

[40] Yichao Zhou, Ying Sheng, Nguyen Vo, Nick Edmonds, and Sandeep Tata. Learning Transferable Node Representations for Attribute Extraction from Web Documents. In K. Selcuk Candan, Huan Liu, Leman Akoglu, Xin Luna Dong, and Jiliang Tang, editors, *WSDM '22: The Fifteenth ACM International Conference on Web Search and Data Mining, Virtual Event / Tempe, AZ, USA, February 21 - 25, 2022*, pages 1479–1487. ACM, 2022. doi:10.1145/3488560.3498424.

[41] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.

Appendix A

Appendix

A.1 Attribute-level performance

Table A.1: Attribute-level performance of our best BERT configuration.

		Global			Instance	
Vertical	Attribute	P	R	F_1	EM	WM
Auto	engine	0.88	0.88	0.88	0.99	1.00
	fuel_economy	0.90	0.90	0.90	0.99	0.99
	model	1.00	1.00	1.00	1.00	1.00
	price	1.00	1.00	1.00	1.00	1.00
Book	author	0.95	0.95	0.95	0.95	0.96
	isbn_13	1.00	1.00	1.00	1.00	1.00
	publication_date	0.89	0.89	0.89	1.00	1.00
	publisher	0.99	0.99	0.99	0.99	0.99
	title	0.98	0.98	0.98	0.98	0.99
Camera	manufacturer	0.99	0.99	0.99	0.99	0.99
	model	0.94	0.94	0.94	0.99	1.00
	price	0.99	0.99	0.99	1.00	1.00
Job	company	0.90	0.90	0.90	1.00	1.00
	date_posted	0.70	0.70	0.70	1.00	1.00
	location	1.00	1.00	1.00	1.00	1.00
	title	1.00	1.00	1.00	1.00	1.00

Continued on next page

Table A.1: Attribute-level performance of our best BERT configuration.

		Global			Instance	
Vertical	Attribute	P	R	F_1	EM	WM
Movie	director	0.93	0.93	0.93	0.93	0.94
	genre	0.91	0.91	0.91	0.91	0.92
	mpaa_rating	0.85	0.85	0.85	0.86	0.86
	title	0.99	0.99	0.99	0.99	0.99
NBA player	height	0.59	0.59	0.59	0.90	0.91
	name	0.80	0.80	0.80	0.99	1.00
	team	0.68	0.68	0.68	0.98	0.98
	weight	0.59	0.59	0.59	0.90	0.90
Restaurant	address	0.87	0.87	0.87	0.97	0.98
	cuisine	0.97	0.97	0.97	0.97	0.97
	name	0.88	0.88	0.88	0.98	0.99
	phone	0.99	0.99	0.99	1.00	1.00
University	name	1.00	1.00	1.00	1.00	1.00
	phone	0.80	0.80	0.80	0.95	0.98
	type	1.00	1.00	1.00	1.00	1.00
	website	0.96	0.96	0.96	1.00	1.00

A.2 Attribute-level performance for different context sizes

Table A.2: Performance of a BERT model for different context sizes. Performance is reported using the instance-level WM score. The best performing context size for a given attribute is marked in bold.

		Context size		
Vertical	Attribute	128	256	512
Auto	engine	1.00	**1.00**	1.00
	fuel_economy	1.00	**1.00**	1.00
	model	1.00	1.00	**1.00**
	price	**1.00**	**1.00**	**1.00**
Book	author	**0.98**	0.98	0.97
	isbn_13	1.00	1.00	**1.00**
	publication_date	**1.00**	**1.00**	1.00
	publisher	1.00	**1.00**	1.00
	title	0.97	0.99	**1.00**
Camera	manufacturer	0.99	0.99	**0.99**
	model	0.99	**1.00**	**1.00**
	price	1.00	0.99	**1.00**
Job	company	0.90	**1.00**	**1.00**
	date_posted	1.00	**1.00**	1.00
	location	1.00	1.00	**1.00**
	title	1.00	1.00	**1.00**
Movie	director	**0.95**	0.94	0.95
	genre	**0.98**	0.88	0.86
	mpaa_rating	0.86	0.84	**0.93**
	title	0.98	0.99	**1.00**
NBA player	height	0.82	0.90	**0.91**
	name	1.00	1.00	**1.00**
	team	**1.00**	**1.00**	**1.00**
	weight	0.79	**0.90**	**0.90**
Restaurant	address	**0.99**	0.83	0.81
	cuisine	**0.98**	0.97	0.96

Continued on next page

Table A.2: Performance of a BERT model for different context sizes. Performance is reported using the instance-level WM score. The best performing context size for a given attribute is marked in bold.

		Context size		
Vertical	Attribute	128	256	512
	`name`	**0.99**	0.99	**0.99**
	`phone`	**1.00**	1.00	1.00
University	`name`	1.00	**1.00**	1.00
	`phone`	0.80	0.98	**0.99**
	`type`	0.99	**1.00**	1.00
	`website`	1.00	1.00	**1.00**

A.3 Attribute-level performance for the zero-shot setting

Table A.3: Performance of our extraction models in the zero-shot setting, per attribute.

		Global			Instance	
Vertical	Attribute	P	R	F_1	EM	WM
Auto	engine	0.42	0.42	0.42	0.42	0.50
	fuel_economy	0.59	0.59	0.59	0.65	0.68
	model	0.61	0.61	0.61	0.61	0.89
	price	0.39	0.39	0.39	0.45	0.56
Book	author	0.76	0.76	0.76	0.76	0.79
	isbn_13	1.00	1.00	1.00	1.00	1.00
	publication_date	0.20	0.20	0.20	0.20	0.39
	publisher	0.38	0.38	0.38	0.38	0.58
	title	0.64	0.64	0.64	0.64	0.69
Camera	manufacturer	0.81	0.81	0.81	0.81	0.89
	model	0.77	0.77	0.77	0.77	0.86
	price	0.36	0.36	0.36	0.36	0.56
Job	company	0.63	0.63	0.63	0.60	0.65
	date_posted	0.29	0.29	0.29	0.41	0.54
	location	0.45	0.45	0.45	0.45	0.65
	title	0.64	0.64	0.64	0.64	0.71
Movie	director	0.76	0.76	0.76	0.76	0.82
	genre	0.75	0.75	0.75	0.75	0.81
	mpaa_rating	0.64	0.64	0.64	0.66	0.67
	title	0.22	0.22	0.22	0.22	0.24
NBA player	height	0.80	0.80	0.80	0.66	0.79
	name	0.68	0.68	0.68	0.68	0.71
	team	0.52	0.52	0.52	0.64	0.74
	weight	0.75	0.75	0.75	0.62	0.80
Restaurant	address	0.27	0.27	0.27	0.26	0.42
	cuisine	0.54	0.54	0.54	0.54	0.60

Continued on next page

Table A.3: Performance of our extraction models in the zero-shot setting, per attribute.

		Global			Instance	
Vertical	Attribute	P	R	F_1	EM	WM
	`name`	0.45	0.45	0.45	0.44	0.46
	`phone`	0.71	0.71	0.71	0.71	0.82
University	`name`	0.72	0.72	0.72	0.72	0.86
	`phone`	0.60	0.60	0.60	0.72	0.73
	`type`	0.69	0.69	0.69	0.69	0.82
	`website`	0.78	0.78	0.78	0.78	0.94

A.4 ClosedIE to OpenIE mapping

Table A.4: ClosedIE to OpenIE mapping for the Movie domain

Website	director	genre	mpaa_rating	title
allmovie	Director	Genres	MPAA Rating	-
amctv	Director:	Genre/Type:	MPAA Rating:	-
hollywood	Director	-	-	-
iheartmovies	Directed by	Genres	MPAA Rating	-
imdb	Director: Directors:	Genres:	Motion Picture Rating	-
metacritic	Director:	Genre(s):	Rating:	-
rottentomatoes	Directed By:	Genre:	Rated:	-
yahoo	Directed by:	Genres:	MPAA Rating:	-

Table A.5: ClosedIE to OpenIE mapping for the NBA player domain

Website	height	name	team	weight
fanhouse	Height	-	Team	Weight
foxsports	Ht	-	-	Wt
espn	Height	-	-	Weight
msnca	Height:	-	Team:	Weight:
si	Height:	-	-	Weight:
slam	Height	-	-	Weight
usatoday	Height:	-	-	Weight:
yahoo	Height	-	-	Weight

Table A.6: ClosedIE to OpenIE mapping for the University domain

Website	name	phone	website	type
collegeprowler	-	-	-	Control:
ecampustours	-	-	-	Affiliation
embark	-	Phone:	-	School Type:
matchcollege	-	General Phone:	Website	Type:
usnews	-	-	Web site:	Institutional Control:

A.5 Failure types

Table A.7: Overview of failure types for randomly sampled failures.

Document	Attribute	True value	Predicted value	Failure type
auto/cars/0431	`engine`	170-hp, 2.5-liter H-4 (regular gas)	170 - hp, 2. 5 - liter h - 4 (regular gas) p2t table p1t tr p0t td 224 - hp, 2. 5 - liter h - 4 (premium	HTML gibberish
auto/cars/0510	`engine`	158-hp, 2.4-liter I-4 (regular gas)	acceleration (2. 4 - liter	Different value
auto/cars/0515	`engine`	310-hp, 4.6-liter V-8 (regular gas)	310 - hp, 4. 6 - liter v - 8 (regular gas) p2t table p1t tr p0t td 381 - hp, 5. 7 - liter v - 8 (flexible ; e85	HTML gibberish
auto/cars/0600	`engine`	182-hp, 2.4-liter I-4 (regular gas)	182 - hp, 2. 4 - liter i - 4 (regular gas) p2t table p1t tr p0t td 264 - hp, 3. 0 - liter v - 6 (flexible ; e85	HTML gibberish
auto/aol/0107	`model`	2011 Subaru Impreza Outback Sport	2011 subaru impreza	Incomplete value
auto/aol/0839	`model`	2010 Mitsubishi Eclipse Spyder	2010 mitsubishi eclipse	Incomplete value
book/barnesandnoble/1544	`author`	Arthur Benjamin	benjamin	Incomplete value
book/bookdepository/1975	`author`	David Else	lonely planet	Different value
book/booksamillion/0118	`author`	J. K. Rowling	scholastic	Different value
book/booksamillion/0788	`author`	John Bomm	sam butcher	Different value
book/booksamillion/1464	`author`	Jane Yolen	jane yolen p2t div p2i content p2c wrap p1t div p1c module p1c details p0t h3 and p2t div p2c module p2c details p1t h3 p0t a mark teague	HTML gibberish
book/booksamillion/1753	`author`	C. W. Randolph	john lee	Different value
book/booksamillion/1793	`author`	Zig Ziglar	napoleon hill	Different value

Continued on next page

Table A.7: Overview of failure types for randomly sampled failures.

Document	Attribute	True value	Predicted value	Failure type
book/booksamillion/1897	author	Robert Louis Stevenson	robert louis stevenson p2t div p2i content p2c wrap p1t div p1c module p1c details p0t h3 and p2t div p2c module p2c details p1t h3 p0t a avi	HTML gibberish
book/buy/1225	author	Agnes F. Vandome	agnes f. vandome p2t table p1t tr p0t td & nbsp ; p2t tr p1t td p0t a john mcbrewster	HTML gibberish
book/christianbook/0225	author	John Haberer	jack haberer	Different value
book/deepdiscount/0469	author	Henry Cloud	henry cloud p2t div p2c col - content p2c clearfix p1t ul p0t li, p2t ul p1t li p0t a lisa guest	HTML gibberish
book/deepdiscount/0739	author	Dr. Anna Klebanov	dr. anna klebanov p2t div p2c col - content p2c clearfix p1t ul p0t li, p2t ul p1t li p0t a dr. john mighton	HTML gibberish
book/deepdiscount/1478	author	Alyssa Shaffer	alyssa shaffer p2t div p2c description - wrap p1t ul p0t li, p2t ul p1t li p0t a chris freytag	HTML gibberish
book/waterstones/0452	author		brady publishing	Nonexisting value
book/waterstones/0942	author	Richard Templar	prentice - hall	Different value
book/waterstones/1092	author		automobile association	Nonexisting value
book/waterstones/1121	author	Joel Rickett	hairy bikers	Different value
book/waterstones/1240	author	Henrietta Thompson	moleskine	Different value
book/waterstones/1309	author	Alexander Games	tommy cooper	Different value
book/waterstones/1694	author	Tsunetomo Yamamoto	kodansha international	Different value
book/barnesandnoble/0058	isbn_13	9781616889364	9781416576617	Different value

Continued on next page

Table A.7: Overview of failure types for randomly sampled failures.

Document	Attribute	True value	Predicted value	Failure type
book/barnesandnoble/0971	isbn_13	9781616802448	9780375975318	Different value
book/abebooks/0674	publication_date		1987	Nonexisting value
book/deepdiscount/0958	publication_date	09/01/2010	08 / 01 / 2010	Different value
book/deepdiscount/0983	publication_date	09/01/2010	08 / 01 / 2010	Different value
book/deepdiscount/1254	publication_date	09/01/2010	08 / 01 / 2010	Different value
book/deepdiscount/1855	publication_date	04/30/2010	07 / 30 / 2010	Different value
book/abebooks/0687	publisher		ballantine books	Nonexisting value
book/abebooks/1195	publisher		pan books	Nonexisting value
book/abebooks/1352	publisher		eden books	Nonexisting value
book/deepdiscount/1168	publisher	Del§ta	del § ta	No error
book/barnesandnoble/0227	title	Mastering the Art of French Cooking Boxed Set	mastering the art of french cooking	Incomplete value
book/barnesandnoble/0285	title	Treasure Island (Barnes & Noble Classics Series)	ambassadors	Different value
book/barnesandnoble/0377	title	The End		Missed attribute
book/barnesandnoble/0762	title	Med-Surg Success		Missed attribute
book/barnesandnoble/1069	title	Eating Well After Weight Loss Surgery		Missed attribute
book/barnesandnoble/1093	title	Forbidden Fantasies	wicked	Different value
book/barnesandnoble/1152	title	Betty Crocker's Diabetes Cookbook		Missed attribute
book/barnesandnoble/1419	title	James Cameron's Avatar	avatar	Incomplete value
book/barnesandnoble/1939	title	Interpretation of Dreams (Barnes & Noble Classics Series)	irma dream	Different value
book/barnesandnoble/1997	title	Forgotten Realms	fourth magic	Different value

Continued on next page

Table A.7: Overview of failure types for randomly sampled failures.

Document	Attribute	True value	Predicted value	Failure type
book/borders/0257	`title`	Twain's The Adventures of Huckleberrry Finn: The Manga Edition	twain ' s the adventures of huckleberrry finn : the manga edition	Incomplete value
book/borders/0271	`title`	The Omnivore's Dilemma: A Natural History of Four Meals	omnivore ' s dilemma : a natural history of four meals	Incomplete value
book/borders/0315	`title`	A Groom of One's Own	groom of one ' s own	Incomplete value
book/borders/0636	`title`	Second Nature: A Gardener's Education	second nature : a gardener ' s education	No error
book/borders/0780	`title`	The Memory Keeper's Daughter	memory keeper ' s daughter	Incomplete value
book/borders/0909	`title`	Somewhere Inside: One Sister's Captivity in North Korea and the Other's Fight to Bring Her Home	somewhere inside : one sister ' s captivity in north korea and the other ' s fight to bring her home	No error
book/borders/1017	`title`	Band of Brothers: E Company, 506th Regiment, 101st Airborne from Normandy to Hitler's Eagle's Nest	band of brothers : e company, 506th regiment, 101st airborne from normandy to hitler ' s eagle ' s nest	No error
book/borders/1085	`title`	Merriam-Webster's Spanish-English Dictionary	merriam - webster ' s spanish - english dictionary	No error
book/borders/1190	`title`	Jack London's The Call of the Wild: The Graphic Novel	jack london ' s the call of the wild : the graphic novel	No error
book/borders/1387	`title`	Coach Wooden's Pyramid of Success	coach wooden ' s pyramid of success	No error
camera/amazon/1073	`manufacturer`	Samsung	canon	Different value
camera/buy/0010	`manufacturer`	OLYMPUS-CAMERAS	olympus	Incomplete value
camera/buy/0261	`manufacturer`	OLYMPUS-CAMERAS	olympus	Incomplete value
camera/buy/0279	`manufacturer`	Memorex	pentax	Different value
camera/buy/0285	`manufacturer`	GENERAL IMAGING COMPANY	ge	Incomplete value

Continued on next page

Table A.7: Overview of failure types for randomly sampled failures.

Document	Attribute	True value	Predicted value	Failure type
camera/buy/0341	`manufacturer`	OLYMPUS-CAMERAS	olympus	Incomplete value
camera/onsale/0012	`manufacturer`	Pentax Imaging	pentax	Incomplete value
camera/onsale/0121	`manufacturer`	Pentax Imaging	pentax	Incomplete value
camera/amazon/0534	`model`	Sony Cyber-shot DSC-TX5 10.2MP CMOS Digital Camera with 4x Zoom with Optical Steady Shot Image Stabilization and 3.5 inch Touch Screen LCD in Silver + 4GB Accessory Kit	sony cyber - shot dsc - tx5 10. 2mp cmos digital camera with 4x zoom with optical steady shot image stabilization and 3. 5 inch touch screen lcd in silver + 4gb accessory kit	Accent mismatch
camera/amazon/0986	`model`	Panasonic Lumix DMC-FZ35 12.1MP Digital Camera w/18x Optical Zoom (Black) BigVALUEInc 8GB Battery/Charger Filter Kit/Lens Accessory Saver Bundle V	panasonic lumix dmc - fz35 12. 1mp digital camera w / 18x optical zoom (black) bigvalueinc 8gb battery / charger filter kit / lens accessory saver bundle	Incomplete value
camera/buy/0047	`model`	Sony DSC-S980 Cyber-shot 12 Megapixel Digital Camera with 4x Optical Zoom, 2.7" LCD, Face Detection & SteadyShot™ Digital Image Stabilization, Black	sony dsc - s980 cyber - shot 12 megapixel digital camera with 4x optical zoom, 2. 7 " lcd, face detection & digital image stabilization, black	Accent mismatch
camera/jr/0130	`price`		149. 99	Nonexisting value
camera/jr/0234	`price`		99. 95	Nonexisting value
job/jobtarget/0214	`company`	Association of American Medical Colleges (AAMC)	aamc	Incomplete value
job/jobtarget/0299	`company`	Tufts University - Medford	tufts university	Incomplete value
job/jobtarget/1231	`company`	Flagstar Bancorp, Inc.	flagstar bank	Incomplete value
job/jobtarget/1602	`company`	Tufts University - Medford	tufts university	Incomplete value

Continued on next page

Table A.7: Overview of failure types for randomly sampled failures.

Document	Attribute	True value	Predicted value	Failure type
job/monster/0149	company	TPI (Tech Providers, Inc.)	unixsabham	Different value
job/monster/0366	company	Collabera	per diem	Different value
job/monster/0604	company	TechExpo Top Secret	cyber27	Different value
job/monster/0684	company	DSCI	apo ae	Different value
job/monster/0749	company	KBW Financial Staffing and Recruiting	unspecified	Different value
job/monster/0944	company	Dell	multiple locations	Different value
job/monster/1193	company	Technical Staffing Specialists Inc	javadevmon	Different value
job/monster/0836	date_posted		date : 11 - 2 - 2010	Nonexisting value
job/careerbuilder/0947	location	US-CT-Hartford		Missed attribute
job/careerbuilder/1432	location	US-AZ-Gilbert		Missed attribute
job/careerbuilder/1436	location	US-CA-Garden Grove		Missed attribute
job/job/1207	location	Union, New Jersey	montvale, new jersey	Different value
job/monster/0684	location	APO AE	543	Nonsense
job/hotjobs/0407	title	> MANAGER, TECHNICAL PROGRAM MANAGEMENT	manager, technical program management	Incomplete value
job/hotjobs/0996	title	> Marcom Specialist	marcom specialist	Incomplete value
job/jobcircle/0256	title	Medical Advisor/ Associate Director 	medical advisor / associate director	Accent mismatch
job/jobtarget/0694	title	Associate FileMaker Developer	differentiators	Different value
job/nettemps/1279	title	Job Title Consultant - Tier 3 / L3 support - 3 positions	consultant - tier 3 / l3 support - 3 positions	Incomplete value
movie/allmovie/0774	director	Agnès Jaoui	agnes jaoui	Accent mismatch
movie/allmovie/0936	director	Pedro Almodóvar	pedro almodovar	Accent mismatch
movie/allmovie/1324	director	Ron Clements	bob goldthwait	Different value

Continued on next page

Table A.7: Overview of failure types for randomly sampled failures.

Document	Attribute	True value	Predicted value	Failure type
movie/allmovie/1855	director	Annabel Jankel	peter hyams	Different value
movie/hollywood/0691	director	François Girard	francois girard	Accent mismatch
movie/iheartmovies/0857	director	Pedro Almod�var	pedro almodvar	Accent mismatch
movie/iheartmovies/1228	director	Gary Weis	gary weis p2t ul p1t li p0t a tamra davis	HTML gibberish
movie/iheartmovies/1482	director	Scott McGehee	naomi foner	Different value
movie/iheartmovies/1902	director	Pedro Almod�var	pedro almodvar	Accent mismatch
movie/imdb/1477	director	Jennifer Flackett	joseph kwong	Different value
movie/metacritic/0104	director	Timothy Bj�rklund	timothy bjrklund	Accent mismatch
movie/metacritic/1712	director	Ishir� Honda	ishir honda	Accent mismatch
movie/msn/0022	director	Barry Poltermann	charles nelson	Different value
movie/msn/0845	director	Paul Barnett	paul barnett p2t div p2c movieinfodirectedbydiv p1t div p1c movieinfodirectorcharacters p0t span, p2t div p2c movieinfodirectorcharacters p1t span p0t a unsu lee	HTML gibberish
movie/rottentomatoes/0304	director	Eric Styles	joseph lees	Different value
movie/rottentomatoes/1129	director	Fr�d�ric Mitterrand	frederic mitterrand	Accent mismatch
movie/yahoo/1097	director	Émile Gaudreault (II)	emile gaudreault (ii	Accent mismatch
movie/yahoo/1109	director		john schmidt	Nonexisting value
movie/yahoo/1635	director		greg germann	Nonexisting value
movie/yahoo/1933	director	Ariel Schulman	andrew jarecki	Different value
movie/allmovie/0902	genre	Horror	horror comedy	Multiple values
movie/iheartmovies/0227	genre	Comedy	comedy p2t ul p1t li p0t a horror	HTML gibberish

Continued on next page

Table A.7: Overview of failure types for randomly sampled failures.

Document	Attribute	True value	Predicted value	Failure type
movie/iheartmovies/0254	genre	Mystery	mystery p2t ul p1t li p0t a thriller	HTML gibberish
movie/iheartmovies/0362	genre	Comedy	comedy p2t ul p1t li p0t a drama	HTML gibberish
movie/iheartmovies/1166	genre	Biography	biography p2t ul p1t li p0t a drama	HTML gibberish
movie/iheartmovies/1700	genre	Drama	crime	Different value
movie/iheartmovies/1919	genre	Horror	musical - horror	Multiple values
movie/msn/0019	genre	Romance	comedy	Different value
movie/msn/0081	genre	Sci-Fi	comedy	Different value
movie/msn/0627	genre	Action	comedy	Different value
movie/msn/0921	genre	Comedy	documentary	Different value
movie/msn/1032	genre	Action	comedy	Different value
movie/rottentomatoes/0301	genre	Art House & International	comedy	Different value
movie/rottentomatoes/1306	genre	Kids & Family	musical	Different value
movie/rottentomatoes/1361	genre	Drama	horror	Different value
movie/rottentomatoes/1381	genre	Drama	documentary	Different value
movie/rottentomatoes/1402	genre	Drama	music	Different value
movie/rottentomatoes/1808	genre	Drama	horror	Different value
movie/rottentomatoes/1983	genre	Mystery & Suspense	horror	Different value
movie/yahoo/1844	genre		animation	Nonexisting value
movie/allmovie/0835	mpaa_rating	P		Missed attribute
movie/allmovie/1466	mpaa_rating	P		Missed attribute
movie/allmovie/1978	mpaa_rating	P		Missed attribute
movie/imdb/0207	mpaa_rating	Rated PG-13 for some sexual content.		Missed attribute
movie/imdb/0328	mpaa_rating	Rated PG for mild language.		Missed attribute

Continued on next page

Table A.7: Overview of failure types for randomly sampled failures.

Document	Attribute	True value	Predicted value	Failure type
movie/imdb/0459	`mpaa_rating`	Rated R for pervasive language and some sexual content.		Missed attribute
movie/imdb/0462	`mpaa_rating`	Rated PG-13 for intense martial arts violence and a scene of sensuality.		Missed attribute
movie/imdb/0491	`mpaa_rating`	Rated R for strong language, some graphic violence and a scene of sexuality.		Missed attribute
movie/imdb/0737	`mpaa_rating`	Rated R for sexual content, nudity, language and a brief violent image.		Missed attribute
movie/imdb/0824	`mpaa_rating`	Rated PG-13 for some innuendo.		Missed attribute
movie/imdb/0839	`mpaa_rating`	Rated R for language, brief strong violence and some sexual images.		Missed attribute
movie/imdb/0873	`mpaa_rating`	Rated R for language.		Missed attribute
movie/imdb/0916	`mpaa_rating`	Rated R for language, some drug content and brief nudity.		Missed attribute
movie/imdb/0957	`mpaa_rating`	Rated R for horror violence, some sexual content and language.		Missed attribute
movie/imdb/1162	`mpaa_rating`	Rated PG-13 for a strong beating and elements of domestic abuse.		Missed attribute
movie/imdb/1221	`mpaa_rating`	Rated PG-13 for mature thematic material including violence, drinking and smoking.	r	Missed attribute
movie/imdb/1633	`mpaa_rating`	Rated PG-13 for intense sequences of frenetic violence and menace, disturbing images and some sensuality.		Missed attribute

Continued on next page

Table A.7: Overview of failure types for randomly sampled failures.

Document	Attribute	True value	Predicted value	Failure type
movie/imdb/1697	`mpaa_rating`	Rated PG for some mild peril and rude humor.		Missed attribute
movie/imdb/1716	`mpaa_rating`	Rated PG-13 for language and sexual content.		Missed attribute
movie/imdb/1814	`mpaa_rating`	Rated R for strong sexual content and language.		Missed attribute
movie/allmovie/0066	`title`	Victor/Victoria	victor	Incomplete value
movie/amctv/0602	`title`	Girl, Interrupted	interrupted	Incomplete value
movie/amctv/0924	`title`	Mon Oncle d'Amérique	mon oncle d'amerique	Accent mismatch
movie/amctv/1433	`title`	Séraphine	seraphine	Accent mismatch
movie/amctv/1446	`title`	Caché	cache	Accent mismatch
movie/hollywood/1092	`title`	Piñero	pinero	Accent mismatch
movie/hollywood/1657	`title`	P2	tomorrow are	Different value
movie/iheartmovies/0026	`title`	A Guide to Recognizing Your Saints (2006)	recognizing your saints (2006	Incomplete value
movie/iheartmovies/0981	`title`	Novios b$\blacksquare$lgaros, Los (2003)	novios blgaros, los (2003	Accent mismatch
movie/iheartmovies/1821	`title`	T$\blacksquare$ky$\blacksquare$ Nagaremono (1966)	tky nagaremono (1966	Accent mismatch
movie/msn/0020	`title`	P2		Missed attribute
movie/rottentomatoes/0106	`title`	Goodbye, 20th Century (Zbogum na dvaesetiot vek) (1999)	goodbye	Incomplete value
movie/rottentomatoes/0207	`title`	H$\blacksquare$ndler der vier Jahreszeiten (The Merchant of Four Seasons)	hndler der vier jahreszeiten (the merchant of four seasons	Accent mismatch
movie/rottentomatoes/0252	`title`	Mad, Sad & Bad (2009)	mad sad & bad	Incomplete value
movie/rottentomatoes/0644	`title`	Venkovsk$\blacksquare$ Ucitel (The Country Teacher)	venkovsk ucitel (the country teacher	Accent mismatch

Continued on next page

Table A.7: Overview of failure types for randomly sampled failures.

Document	Attribute	True value	Predicted value	Failure type
movie/rottentomatoes/1114	`title`	Sex and Lucia (Luc  a y el sexo)	sex and lucia (luca y el sexo	Accent mismatch
movie/rottentomatoes/1732	`title`	L' Amour l'Apr  s-Midi (Chloe in the Afternoon) (Love in the Afternoon)	l'amour l'aprs - midi (chloe in the afternoon) (love in the afternoon	Accent mismatch
nbaplayer/foxsports/0002	`height`	6' 5"		Missed attribute
nbaplayer/foxsports/0037	`height`	6' 9"		Missed attribute
nbaplayer/foxsports/0046	`height`	6' 10"		Missed attribute
nbaplayer/foxsports/0054	`height`	6' 10"		Missed attribute
nbaplayer/foxsports/0067	`height`	7' 0"		Missed attribute
nbaplayer/foxsports/0084	`height`	6' 2"		Missed attribute
nbaplayer/foxsports/0087	`height`	6' 7"	6. 1	Different value
nbaplayer/foxsports/0125	`height`	6' 10"		Missed attribute
nbaplayer/foxsports/0153	`height`	7' 1"		Missed attribute
nbaplayer/foxsports/0183	`height`	7' 1"		Missed attribute
nbaplayer/foxsports/0186	`height`	6' 10"		Missed attribute
nbaplayer/foxsports/0188	`height`	6' 6"	6. 0	Different value
nbaplayer/foxsports/0189	`height`	6' 9"		Missed attribute
nbaplayer/foxsports/0231	`height`	6' 9"		Missed attribute
nbaplayer/foxsports/0251	`height`	7' 0"		Missed attribute
nbaplayer/foxsports/0313	`height`	6' 11"		Missed attribute
nbaplayer/foxsports/0334	`height`	6' 10"		Missed attribute
nbaplayer/foxsports/0340	`height`	6' 7"		Missed attribute

Continued on next page

Table A.7: Overview of failure types for randomly sampled failures.

Document	Attribute	True value	Predicted value	Failure type
nbaplayer/foxsports/0414	height	6’ 8"	6. 2 p2t table p1t tr p0t td league average p2t table p1t tr p0t td 8. 2 p2t table p1t tr p0t td league leader p2t table p1t tr p0t td 27. 4	HTML gibberish
nbaplayer/foxsports/0419	height	6’ 7"		Missed attribute
nbaplayer/usatoday/0142	name	Jeremy Lin	jeremey lin	Spelling mismatch
nbaplayer/wiki/0269	name	Emanuel Ginóbili	emanuel ginobili	Accent mismatch
nbaplayer/wiki/0407	name	Donté Greene	donte greene	Accent mismatch
nbaplayer/foxsports/0002	weight	200		Missed attribute
nbaplayer/foxsports/0020	weight	245		Missed attribute
nbaplayer/foxsports/0046	weight	240		Missed attribute
nbaplayer/foxsports/0054	weight	255		Missed attribute
nbaplayer/foxsports/0067	weight	235		Missed attribute
nbaplayer/foxsports/0071	weight	210		Missed attribute
nbaplayer/foxsports/0125	weight	240		Missed attribute
nbaplayer/foxsports/0183	weight	250		Missed attribute
nbaplayer/foxsports/0188	weight	230		Missed attribute
nbaplayer/foxsports/0189	weight	225		Missed attribute
nbaplayer/foxsports/0231	weight	210		Missed attribute
nbaplayer/foxsports/0280	weight	175		Missed attribute
nbaplayer/foxsports/0313	weight	246		Missed attribute
nbaplayer/foxsports/0325	weight	259		Missed attribute
nbaplayer/foxsports/0334	weight	240		Missed attribute
nbaplayer/foxsports/0340	weight	215		Missed attribute
nbaplayer/foxsports/0352	weight	265		Missed attribute

Continued on next page

Table A.7: Overview of failure types for randomly sampled failures.

Document	Attribute	True value	Predicted value	Failure type
nbaplayer/foxsports/0355	weight	207		Missed attribute
nbaplayer/foxsports/0395	weight	260		Missed attribute
nbaplayer/foxsports/0419	weight	230		Missed attribute
restaurant/opentable/1523	address	Boston, MA 02116	boston	Incomplete value
restaurant/restaurantica/0561	address	101 S New York St	al	Nonsense
restaurant/restaurantica/0653	address	1508 Highway 62	al	Nonsense
restaurant/restaurantica/1225	address	569 N 2nd St	al	Nonsense
restaurant/restaurantica/1259	address	815 Indianapolis Ave	al	Nonsense
restaurant/restaurantica/1657	address	3230 23rd Ave	al	Nonsense
restaurant/restaurantica/1705	address	928 Grand Ave	al	Nonsense
restaurant/restaurantica/1749	address	710 Harrison Ave	al	Nonsense
restaurant/restaurantica/1890	address	136 Simsbury Rd	al	Nonsense
restaurant/restaurantica/1996	address	328 Pemberwick Road	al	Nonsense
restaurant/urbanspoon/0203	address	2764 Aurora Ave	chicago	Different value
restaurant/urbanspoon/0538	address	164 Old Mill Ave	knoxville	Different value
restaurant/urbanspoon/0561	address	223 Jackson Sq	knoxville	Different value
restaurant/urbanspoon/0931	address	1051 Conestoga Rd	philadelphia	Different value
restaurant/urbanspoon/1190	address	121 Park Ln	seattle	Different value
restaurant/urbanspoon/1816	address	63 Church Street	birmingham	Different value
restaurant/urbanspoon/1862	address	333 Littleton Rd	boston	Different value
restaurant/usdiners/1467	address	1837 W GUADALUPE RD,	phoenix	Different value
restaurant/usdiners/1571	address	Naples, FL 34102	naples	Incomplete value
restaurant/usdiners/1705	address	2533 Constance St	2533 constance st p2t tr p1t td p0t span p0c news (3rd st	HTML gibberish

Continued on next page

Table A.7: Overview of failure types for randomly sampled failures.

Document	Attribute	True value	Predicted value	Failure type
restaurant/gayot/0139	cuisine	French	french bistro	Incomplete value
restaurant/gayot/0987	cuisine	American	california	Different value
restaurant/gayot/1189	cuisine	Italian	cafe	Different value
restaurant/gayot/1423	cuisine	Mediterranean	tapas	Different value
restaurant/gayot/1517	cuisine	Café	eclectic	Different value
restaurant/gayot/1740	cuisine	California	small plates	Different value
restaurant/gayot/1743	cuisine	Brewpub	pub	Incomplete value
restaurant/gayot/1822	cuisine	Seafood	hot plates	Different value
restaurant/gayot/1829	cuisine	French	georgian	Different value
restaurant/gayot/1884	cuisine	American	cafe	Different value
restaurant/restaurantica/0870	cuisine	Asian	buffet	Different value
restaurant/restaurantica/0873	cuisine	Asian	buffet	Different value
restaurant/restaurantica/0953	cuisine	Asian	mexican	Different value
restaurant/urbanspoon/0102	cuisine	American	pizza	Different value
restaurant/urbanspoon/0401	cuisine	Pub Food	irish	Different value
restaurant/urbanspoon/0666	cuisine	Sandwiches/Subs	italian	Different value
restaurant/urbanspoon/0753	cuisine	Bakery	pizza	Different value
restaurant/urbanspoon/0951	cuisine	Wine Bar	french	Different value
restaurant/urbanspoon/1819	cuisine	Bakery	japanese	Different value
restaurant/urbanspoon/1862	cuisine	Burgers	chicken	Different value
restaurant/gayot/0094	name	Gilly's	gilly ' s	No error
restaurant/gayot/0332	name	BC Café	bc cafe	Accent mismatch
restaurant/gayot/0430	name	Fisher's Café & Pub	fisher's cafe & pub	Accent mismatch
restaurant/gayot/0543	name	Café on the Green	cafe on the green	Accent mismatch

Continued on next page

Table A.7: Overview of failure types for randomly sampled failures.

Document	Attribute	True value	Predicted value	Failure type
restaurant/gayot/0967	name	Bouchée	bouchee	Accent mismatch
restaurant/gayot/1085	name	Café Greek	cafe greek	Accent mismatch
restaurant/gayot/1189	name	Oliveto Café	oliveto cafe	Accent mismatch
restaurant/gayot/1410	name	Café Metropol	cafe metropol	Accent mismatch
restaurant/gayot/1535	name	Derek's	derek ' s	No error
restaurant/gayot/1543	name	Eddie V's Prime Seafood	eddie v ' s prime seafood	No error
restaurant/gayot/1884	name	Wilde Roast Café	wilde roast cafe	Accent mismatch
restaurant/gayot/1955	name	New Orleans Cake Café & Bakery	new orleans cake cafe & bakery	Accent mismatch
restaurant/opentable/0139	name	Generations Restaurant	generations	Incomplete value
restaurant/urbanspoon/0181	name	Amélie's French Bakery & Cafe	amelie's french bakery & amp ; cafe	Accent mismatch
restaurant/urbanspoon/0542	name	Captain Ernie's Fish House Restaurant	captain ernie ' s fish house restaurant	No error
restaurant/urbanspoon/0758	name	Bogotá Latin Bistro	bogota latin bistro	Accent mismatch
restaurant/urbanspoon/1793	name	Golden Temple Natural Grocery & Café	golden temple natural grocery & amp ; cafe	Accent mismatch
restaurant/usdiners/0339	name	Café Polonia	cafe polonia	Accent mismatch
restaurant/zagat/1312	name	mar'sel	mar ' sel	No error
restaurant/zagat/1370	name	Café Prima Pasta	cafe prima pasta	Accent mismatch
restaurant/restaurantica/0917	phone	9 (280) 632-1388	280) 632 - 1388	Incomplete value
university/embark/0060	phone	480 545-8755	480 926 - 1371	Different value
university/embark/0147	phone	205-226-4696	205 - 226 - 3074	Different value
university/embark/0237	phone	888-227-3552	612 - 339 - 8022	Different value
university/embark/0313	phone	216 241-2930	216 241 - 5432	Different value
university/embark/0314	phone	440 473-6273	440 473 - 0530	Different value

Continued on next page

Table A.7: Overview of failure types for randomly sampled failures.

Document	Attribute	True value	Predicted value	Failure type
university/embark/0826	`phone`	616 965-3931	616 965 - 8850	Different value
university/embark/0852	`phone`	352 787-3747	352 365 - 3501	Different value
university/embark/0856	`phone`	409 983-4921	409 984 - 6000	Different value
university/embark/0945	`phone`	310-338-2848	310 - 338 - 2899	Different value
university/embark/0973	`phone`	617 595-6768	617 595 - 3560	Different value
university/embark/0981	`phone`	570-348-6234	570 - 961 - 4763	Different value
university/embark/1061	`phone`	989 328-2111	989 328 - 2950	Different value
university/embark/1394	`phone`	559 448-8282	559 448 - 8250	Different value
university/embark/1438	`phone`	707-664-2778	707 - 664 - 2060	Different value
university/embark/1562	`phone`	423 892-9882	423 892 - 5006	Different value
university/embark/1796	`phone`	940-565-2681	940 - 565 - 2408	Different value
university/studentaid/0177	`phone`	800 968-6920 (toll free)	719 336 - 2248 p2t tr p1t td p0t b 800 968 - 6920 (toll free	HTML gibberish
university/studentaid/0347	`phone`	800 MBBC-WIS (toll free)	920 261 - 9300 p2t tr p1t td p0t b 800 mbbc - wis (toll free	HTML gibberish
university/studentaid/0962	`phone`	800 460-1328 (toll free)	214 333 - 7100 p2t tr p1t td p0t b 800 460 - 1328 (toll free	HTML gibberish
university/studentaid/1331	`phone`	800 300-7420 (toll free)	209 667 - 3122 p2t tr p1t td p0t b 800 300 - 7420 (toll free	HTML gibberish
university/embark/1035	`website`	273 E. Erie Street		Missed attribute
university/matchcollege/1421	`website`	http:		Missed attribute

A.6 Attribute-level mean reciprocal rank of correct predictions

Table A.8: Attribute-level mean reciprocal rank (MRR) of correct segment predictions.

Vertical	Attribute	MRR
Auto	engine	1.00
	fuel_economy	1.00
	model	1.00
	price	1.00
Book	author	0.98
	isbn_13	1.00
	publication_date	1.00
	publisher	1.00
	title	1.00
Camera	manufacturer	1.00
	model	1.00
	price	1.00
Job	company	0.99
	date_posted	1.00
	location	1.00
	title	1.00
Movie	director	0.98
	genre	0.96
	mpaa_rating	1.00
	title	0.99
NBA player	height	1.00
	name	1.00
	team	0.98
	weight	1.00
Restaurant	address	0.97
	cuisine	0.98
	name	1.00

Continued on next page

Table A.8: Attribute-level mean reciprocal rank (MRR) of correct segment predictions.

Vertical	Attribute	MRR
	`phone`	1.00
University	`name`	1.00
	`phone`	0.99
	`type`	1.00
	`website`	1.00