Transfer learning for multilingual vacancy text generation

Anna Lőrincz
University of Amsterdam
anna@lorincz.org

David Graus
Randstad Groep Nederland
david.graus@randstadgroep.nl

Dor Lavi
Meta
dorlavi@meta.com

João L. M. Pereira
University of Amsterdam
INESC-ID and IST, Universidade de Lisboa
j.p.pereira@uva.nl

Abstract

Writing job vacancies can be a repetitive and expensive task for humans. This research focuses on automatically generating parts of vacancy texts, i.e., the benefits section, given structured job attributes as input using mT5, the multilingual version of the state-of-the-art T5 transformer model. While transformers are accurate at generating coherent text, they can struggle with correctly including structured (input) data in the generated text. Including this input data correctly is crucial for vacancy text generation; otherwise, job seekers may be misled. To evaluate how the model includes the different types of structured input, we propose a novel domain-specific metric: 'input generation accuracy'. Our metric aims to address the shortcomings of Relation Generation, a commonly used evaluation metric for data-to-text generation that relies on string matching, as our task includes evaluating generated texts based on binary and categorical inputs. Using our novel evaluation method, we measure how well the input is included in the generated text separately for different types of inputs (binary, categorical, numeric), offering another contribution to the field. In addition, we evaluate how accurately the mT5 model generates texts in the requested languages. Our experiments show that mT5 is highly accurate at generating texts in the correct (requested) languages, and at handling seen categorical and binary inputs correctly. However, mT5 performed worse when generating text from unseen city names or working with numeric inputs.

1 Introduction

Integrating Natural Language Processing (NLP) solutions to improve the human workforce offers great opportunities for process automation (Devarajan, 2018). Such process automation can be to automate the writing of parts of the vacancies using the given structured data. This research was carried out with Randstad Netherlands, a Dutch HR services company that provides vacancies in Dutch and English.

Natural Language Generation (NLG) is the subfield of NLP focusing on automatic text writing. Text generation tasks can be categorized depending on the input into text-to-text generation (e.g., text translation) and data-to-text generation (e.g., the generation of Wikipedia biographies). Generating vacancy text falls under the data-to-text generation category, since particular job-specific information (for example: `MINSALARY = 2500, MAXSALARY = 2700, SALARYTYPE = 'per month'`) is provided as input and a coherent sentence that combines the given information must be generated as the output (for example *'You will receive between 2500 and 2700 euros on a monthly bases'*). Early data-to-text generation approaches used hand-engineered templates (Kukich, 1983; McKeown, 1992; McRoy et al., 2000) to generate texts. These templates can be intuitively explained as a series of written text elements, with certain segments serving as the template's 'backbone' and the remaining segments being filled up with information from the structured data (Wang and Cardie, 2013). From the previous example: *'You will receive between 2500 and 2700 euros on a monthly bases'* the following template can be created *'You will receive between <MINSALARY> and <MAXSALARY> euros on a monthly bases'*. Such simple (template-based) method was used as the baseline for this research. While this solution is computationally cheap, fast and the inputs will be correctly included, a lot of manual effort is required to create the templates and the generated texts will be very repetitive.

Transformer models are a type of neural network architecture that use transfer learning. Transfer learning is the technique where these models are initially trained (pre-trained) on a large unlabeled text dataset and some supervised tasks. The 'knowledge' learned through pre-training is then reused on the new task, making transformer methods cur-

rently the most effective solution for text generation (Devlin et al., 2019). While there is no transformer model available that was pre-trained on data-to-text generation tasks, recent research in (Kale and Rastogi, 2020) showed that transformer models - that were pre-trained on text-to-text generations tasks - outperform traditional (non-transformer) neural network solutions for data-to-text generation task after fine-tuning. Based on this finding and the fact that our dataset is multilingual, we used the multilingual version of T5, mT5 (Xue et al., 2021). In contrast to the template-based method, transformers can provide a wide range of text variation. However, when generating text, transformers pick the next word with some randomness, thus they are less accurate at including the input correctly.

This paper investigates how the benefit sections of vacancies (namely: *salary, hours, contract and location*) can be generated using mT5. Our contribution is three-fold. On one hand, we use mT5 in a new domain, with a relatively small and proprietary dataset - that contains many noise and incorrect samples - in a multilingual setting. On the other hand, we confirm the importance of domain specific evaluation metrics in data-to-text generation, by analyzing - with the metrics developed by us - how accurate mT5 is in terms of including the input correctly in the generated text. We also investigate how creating more training data (through translation and generating synthetic samples) can improve this accuracy. Focusing on including the input correctly is particularly important in this domain, where incorrect wages, or stating hourly salaries instead of monthly can be misleading to candidates.

This paper focuses on answering the following research question:

RQ1 To what extent can the state-of-the-art multilingual text-to-text generation transformer, mT5 (Xue et al., 2021) be used to generate the benefit sections of vacancies in a multilinguistic environment?

To answer this research question we aim to answer the following sub-questions:

RQ1.1 How accurate is mT5 in terms of generating the text in the correct language for the benefit sections of vacancies?

RQ1.2 When using new domain specific metrics to measure the input generation accuracy, how does mT5 perform, in terms of including different types of inputs (numerical, binary, categorical) in the generated text?

RQ1.3 How can creating extra training data by translating the training samples automatically and generating synthetic training data increase the accuracy of the model regarding language and input generation?

2 Related Work

The current state-of-the art solutions in text generation are transformer models that rely on transfer learning, where a model is initially pre-trained on a large dataset, before being fine-tuned on a downstream task (for example text summarization). Pre-training on a large, unlabelled dataset is important for transformer models, because during this pre-training the model learns the basics of language. T5 transformer was proposed with a new dataset by Google Raffel et al. (2020). This so called Colossal Clean Crawled Corpus (C4)[*] is a thoroughly cleaned massive text dataset, scraped from many websites for pre-training the model. The model is classified as text-to-text, since both the input and output are always text strings. To define which task (for example text summarization) the model should perform, a task-specific prefix was appended to the input during the pre-training. This prefix is simply a short synopsis of the task that is subsequently used throughout fine-tuning to teach the model a new task. This same prefix is then used to specify what task to perform when employing the model. Such a prefix may be 'Translate English to German:' (for machine translation) or 'Summarize' (for text summarization) (Raffel et al., 2020). Thanks to this task-specific prefix and the large pre-training dataset, the model is adaptable enough to be fine-tuned on different downstream tasks with the same loss function and hyperparameters.

2.1 Transformer-based data-to-text generation application domains

All relevant transformer based data-to-text research have been conducted in three domains (sport game summaries, open and closed domain Wikipedia). The researches in the Wikipedia domain use ToTTo (Parikh et al., 2020) and WebNLG [†] (Gardent et al., 2017; Zhou and Lampouras, 2020; Castro Ferreira et al., 2020) datasets with training sizes of 120K

[*] https://www.tensorflow.org/datasets/catalog/c4
[†] https://webnlg-challenge.loria.fr/challenge_2020

and 25.3K. The best performing model for the ToTTo dataset is a simple fine-tuned T5 model (Kale and Rastogi, 2020). For the WebNLG domain the best one is T5 with some adjustment to control the generation by including conditional, input-dependent information in the prefixes (Clive et al., 2021). This domain has received more attention than the others for multilingual data-to-text generation (Moussallem et al., 2020). Agarwal et al. (2020) found that by fine-tuning the model in two languages via machine translation (on the WebNLG dataset), a bilingual T5 model can outperform two separate monolingual T5 models. We use an equivalent strategy for enhancing the multilingual model by automatically translating the training samples.

The last domain - sport game summaries - uses the RotoWire dataset (Wiseman et al., 2017) with 4.9K training samples to generate NBA basketball game summaries from the structured game scores. An intriguing transformer-based approach (Gong et al., 2019) demonstrated how producing syntactic training data by replacing the scores with new, synthetic values improves performance. We use the same concept in this paper.

Recently, Qin et al. (2022) proposed a neural approach to generate the requirements section of the job description. This research however is not a data-to-text generation study, because it focuses on generating the requirements for jobs (such as the required skills), based on the fluent text of the tasks section of the job descriptions. Our research concentrates on a different task within the domain by generating the benefit section of the vacancies. We use a transformer model and a data-to-text approach on a bilingual dataset, which is highly unbalanced (having about 95% of the samples in Dutch) and diverse regarding the data types (numeric, categorical, binary), which sets the research apart from previous work.

2.2 Existing evaluation methods

Typical quality of text generation metrics are Bilingual Evaluation Understudy (BLEU) score (Papineni et al., 2002), Recall-Oriented Understudy for Gisting Evaluation (ROUGE) score (Lin, 2004) and Metric for Evaluation of Translation with Explicit ORdering (METEOR) score (Lavie and Agarwal, 2007). ROUGE measures the recall by concentrating on how many n-grams of the reference (human written text) appeared in the generated text, whereas BLEU measures precision of the appearing n-grams by measuring on how much of the output text appeared in the reference (Lin, 2004). METEOR was proposed for machine translation to improve the alignment with human judgement. The metric computes the explicit word-to-word match between the output and the reference, however it has many properties that are missing from the previous metrics. METEOR performs stemming (using the Porter Stemmer) and also takes synonyms into consideration (Lavie and Agarwal, 2007). The traditional quality of text evaluation methods however, fail to evaluate the generated text in terms of their fidelity to the input data (Wiseman et al., 2017). To solve this issue Relation Generation (RG) metric was proposed in (Wiseman et al., 2017). The metric measures how well the system includes the input, however it assumes that the entities can be detected in the generated text using string matching (Dhingra et al., 2019). This is not the case for our task, because while some entities can be detected: like the `location` (the city name), numbers are more problematic due to the monetary units. For example, in the Netherlands the decimal point is indicated by a comma instead of a period. Lastly, measuring the binary variable (`hidesalary`) is not in any way detectable by string matching only. Therefore, we propose new rule based domain specific measures, which we describe in subsection 3.6.

3 Methodology

3.1 Dataset

The internal dataset of Randstad Netherlands was used for the research which contains both structured vacancy data (e.g., salary amount), which we use as input for text generation, and the corresponding human written vacancy texts (`targettext`), i.e., the desired output of the text generation. We focused on the following four benefit sections: *salary, location, hours and contract*. Figure 1 shows an example for each of these benefit sections, using a vacancy for a Customer Service Representative published on Randstad Netherlands' website[‡]. Table 1 shows the 11 different input fields and how each benefit section uses different ones (except the language, which is used for every section). Four data types are present in the dataset: integer (`min/maxhours`) and non-integer (`min/maxsalary`) numericals, categoricals

[‡] https://www.randstad.com/find-a-job/

(location, contracttype, salarytype) and binary (hidesalary, fixhour, fixsalary).

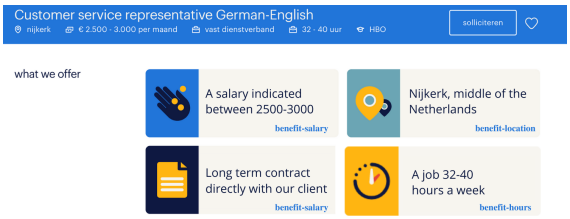

Figure 1: The figure illustrates the layout of the benefit section on the website of Randstad Netherlands.

3.2 Template-based method

When designed properly, template-based models always perform perfect in terms of including the inputs and using the correct language. We implemented a template-based method to compare the performance of mT5 and to provide an alternative solution to the transformer approach for this domain. The pipeline of this method starts by creating a template of every `targettext` from the training data, by replacing the numeric and the `location` input values in the text with a token corresponding to the feature name. (For example from the sample '€2800 - €3000 gross per month' we create a template as ' €<MINSALARY> - €<MAXSALARY> gross per month'.) Next, we choose a set of templates for each combination of categorical data by selecting the top five most common templates for each combination from the training data. These previous steps are performed only once, whereas the next step, the text generation, is performed every time a new benefit section needs to be generated with the method. As part of this text generation step, for each sample a template is picked from the correct combination and then, this picked template is filled with the correct values by replacing the feature tags with their values. The steps are the same for every section and every combination of categories has five templates. Appendix A contains an example for the template-based method.

3.3 mT5 model

We used hugging face's PyTorch implementation[§] with the default tokenizer and default loss (cross-entropy loss) to fine-tune the base version of the mT5 model. For optimizer we used Adafactor with the recommended setting by Shazeer and Stern (2018). Similar to the task-specific prefix, the input for the mT5 model is provided in a text format. For data-to-text generation, the input is typically provided by first including the feature name and then the feature value. Because transformer models are extremely sensitive to how the input is given (Lester et al., 2021), we experimented for each section with three distinct input formats during the fine-tuning of the model. The only difference between the inputs is how the feature name is represented in the input. For each section the 'FEATURENAME' token was changed to the feature name which we want to include and the 'value' token was replaced by the value of the feature. For each of our three formats, listed below, we present an example from the location benefit section, where our only feature is the `location` and the value of the feature is 'Amsterdam'.

- **input A:** `<FEATURENAME> value`
  example: *`<LOCATION> Amsterdam`*
- **input B:** `FEATURENAME = value`
  example: *`LOCATION = Amsterdam`*
- **input C:** `<FEATURENAME> value </FEATURENAME>`
  example: *`<LOCATION> Amsterdam </LOCATION>`*

For English text generation we used the prefix 'Generate in English' and for Dutch 'Generate in Dutch'. We specified the prefix and the input in English.

We fine-tuned the original dataset using each input format and choose the best performing one based on the input generation accuracy score (the newly developed metrics explained in subsection 3.6). When fine-tuning the models on the translated and the synthetic dataset we used the same input and prefix as on the original dataset. Our hyperparameter tuning consisted of finding the number of epochs that had the highest input generation accuracy. For the generation, we set the sampling parameter to `True`. Allowing sampling means that the next word is randomly picked from the conditional probability distribution, thus the model is more diverse and does not generate the same sentence for the same input, unless the random seed is identical. This is important, because otherwise all samples with the same input would have the same generated text, making the outputs very repetitive (for example every job that has 40 working hours, which is quite common). For the text generation, we also set the 'top_p' parameter to 1 to ensure that only the tokens whose combined probability adds

[§] https://huggingface.co/docs/transformers/model_doc/mt5

Relevant benefit section	Feature	Description	Data type	Occurring values
All	language	Language of the vacancy text	categorical (string)	{English, Dutch}
Salary	hidesalary	Whether salary amount should be mentioned	binary	{True, False}
	minsalary	Lower limit of salary amount in euro	numeric (float)	9.63 - 41.0 for hourly, 250 - 4500 for monthly
	maxsalary	Upper limit of salary amount in euro	numeric (float)	9.7 - 45.0 for hourly, 250 - 6000 for monthly
	fixsalary	Whether minsalary and maxsalary are equal	binary	{True, False}
	salarytype	Frequency of payment	categorical (string)	{Monthly, Hourly}
Hours	minhour	Minimum working hours per week	numeric (int)	2 - 55
	maxhour	Maximum working hours per week	numeric (int)	2 - 55
	fixhour	Whether minhour and maxhour are equal	binary	{True, False}
Contract	contracttype	Type of offered contract	categorical (string)	{Temporary, Temporary with a possibility of fixed, Fixed}
Location	location	City name of work location	categorical (string)	-

Table 1: Overview of the features in the dataset

up to 100% are kept for generation. Additionally, we set 'top_k' parameter to 10 to make sure to only choose from the ten most probable tokens.

3.4 Generating extra training data

We applied two methods to generate extra training data. One for producing additional examples in each language (inspired by Agarwal et al. (2020)) by translating all English training samples to Dutch and all Dutch training samples to English using the Googletrans library[¶]. Then, we repeated the pre-processing steps, removed the duplicated samples and appended this new training set to our original training set, approximately doubling our training set and obtaining a balanced dataset in terms of language. We refer to this training set as the translated training set. For the second method (similarly to Gong et al. (2019)), we generated synthetic data for the sections with the numeric inputs (salary and hours). We guaranteed that the range between the lower and higher values (lower and higher bounds) were not exceeded (for working hours and monthly and hourly salary amount separately). After randomly selecting the new numbers, we replaced the original numeric values in the targettext to create the new synthetic training samples. We rounded every value to two decimals and ensured using the correct monetary unit for the salary section. For the hours section, we only picked from integer numbers, following the nature of the training data. We created such synthetic training samples by replacing values three times on every training sample from the translated dataset. After creating the synthetic data we performed the pre-processing steps and removed any duplicates, thus we received an approximately three times larger, balanced - regarding the language - dataset that included some randomness and covered more numeric values than our previous training sets. We then added this training set to our translated dataset, which we refer to as synthetic training data in the rest of the paper.

[¶] https://pypi.org/project/googletrans/

3.5 Experimental setup

We split the data for each benefit section into training, validation and test sets with a 60-20-20% ratio. Table 9 in Appendix C contains the sample sizes for each benefit sections. The details about how the random seed was set for different parts of the workflow in order to ensure reproducibility and fair comparison between models can be found in Appendix D.

3.6 Evaluation metrics

3.6.1 Text Quality

To measure the quality of text, we use the BLEU-1, BLEU-2, ROUGE-1, ROUGE-2 and METEOR scores (previously explained in section 2). We only use the BLEU and ROUGE scores up to two n-grams because our sections are rather short (5.5 to 7 words on average). Additionally, we record the size of vocabulary by automatically counting the number of unique words in the generated texts after removing all numeric values, and excluding the city name for the location section. This way we ensure that including a wrong input does not change the size of vocabulary.

3.6.2 Language accuracy

A main goal of this study is to determine, whether the mT5 model is reliable in terms of generating

text in the requested language. Thus, we compare the requested language to the language of the generated text. We used the Lingua library[‖] for language detection, as this library is suitable for detecting language of short sentences specifically. To determine the language accuracy we calculate the percentage for each language where the detected language matches the intended one.

3.6.3 Input generation accuracy

For the newly developed 'input generation accuracy' metrics, we calculate the number of correct samples divided by the number of all samples. The correct samples are identified with predefined rules with string matching. For the location section a sample is considered correct if the city name is correctly included. For the contract section, we use a collection of terms to classify the created text into one of three potential categories (Temporary, Temporary with a possibility of fixed, Fixed). For the hours and salary sections we extract the numbers from the generated text and compare them to the input numbers. If there is any difference between the two set of numbers we treat the sample incorrect. Furthermore we check whether each number is only included once. This is especially crucial for the fix salaries otherwise for example the following text would be considered correct: *'You will work between 40 and 40 hours'*. For the salary section, we also check whether the monetary unit is set properly (the decimal is separated with a comma) and the rounding is correct (all integers are rounded to whole numbers and non-integers to two decimals). Additionally, we use string matching to assess the categorical `salarytype` input generation accuracy. First, we lowercase the generated text and then for example if the salary type is 'per month', we consider the sample correct if the generated text contains 'month' but not 'hour'. As a result, records are not only penalized if they include the wrong salary type, but also if they do not contain the correct one. Finally, we measure the binary input generation accuracy (for `hidesalary`) by detecting whether any numeric values appear in the generated text.

4 Results

Table 2 shows the results for each section. It shows the input format *(Input)*, epoch size *(Epoch)*, quality of text scores *(METEOR, BLEU-1, BLEU-2, ROUGE-1, ROUGE-2)*, language accuracy scores *(English, Dutch)*, input generation accuracy *(for categorical, numerical and binary inputs)*, and vocabulary size in words.

4.1 Location

Rows 1-3 in Table 2 show the results obtained for the location benefit section. The best performing input format for mT5 (based on input generation accuracy) was *'FEATURE = value'* (input B). A perfect language accuracy was already achieved on the original dataset (row 2). By fine-tuning the model on the translated dataset, we were able to increase the input generation accuracy to 84.87% (row 3). mT5 fine-tuned on this translated data (row 3) reached similar results regarding the METEOR score and slightly outperformed the template-based method (row 1) in terms of BLEU-2 and ROUGE-2 score. Additionally, the size of vocabulary was more than twice (315-339) for the generated text when compared to the template-based method (129).

4.2 Contract

As mentioned in section 3, we used the English prefix value of the input when fine-tuning the models. However, in the case of the contract section the English language accuracy remained zero even after using the translated dataset for fine-tuning (row 6 on Table 2). Then we fine-tuned the model using a translated version of the prefix and the input. For example for an English sample the prefix was given as *'Generate in English'* and the input format was *'CONTRACTTYPE = fix'*, while for a Dutch sample the prefix was specified as *'Genereren in het Nederlands'* and the input was translated to Dutch and given as *'CONTRACTTYPE = vast'*. As shown on row 7, with the translated prefix and input, the model was able to achieve high language accuracy for English (92.31%) and a high input generation accuracy (96.44%) too. Adding the translated data (row 8), however was found unnecessary as it resulted in a decrease in the categorical input generation accuracy (to 74.19%). Overall, the transformer model achieved similar results to the template-based method (row 4) regarding the language and the input accuracy, while under performing in terms of the quality of text scores. Furthermore, the used vocabulary by the transformer had a 3.5 times larger size (170-263) than the template-based method (77).

[‖] https://pypi.org/project/lingua/

	Benefit section	Model	Training data	Input	Epochs	METEOR	BLEU-1	BLEU-2	ROUGE-1	ROUGE-2	English lang. acc.	Dutch lang. acc.	Binary input gen. acc.	Categorical input gen. acc.	Numeric input gen. acc.	Size of voc.
1	Location	Template-based	Original	-	-	19.95	**30.24**	11.84	**37.67**	10.62	**100.00**	**100.00**	NA	**100.00**	NA	129
2	Location	mT5	Original	B	40	**20.88**	28.35	**13.53**	34.43	**13.09**	**100.00**	**100.00**	NA	76.97	NA	315
3	Location	mT5	Translated	B	50	20.33	27.77	13.40	32.36	11.87	**100.00**	**100.00**	NA	84.87	NA	**339**
4	Contract	Template-based	Original	-	-	**37.16**	**36.88**	**25.93**	**56.03**	**30.63**	**100.00**	**100.00**	NA	**100.00**	NA	77
5	Contract	mT5	Original	C	15	25.43	26.42	14.43	34.81	15.12	0.00	99.54	NA	78.42	NA	219
6	Contract	mT5	Trans.	C	20	25.09	26.55	14.10	34.02	14.33	0.00	99.77	NA	66.63	NA	170
7	Contract	mT5	Original	B (tran.)	15	29.10	29.42	16.51	39.14	16.88	92.31	98.97	NA	96.44	NA	**263**
8	Contract	mT5	Translated	B (tran.)	15	26.71	27.37	15.02	34.85	15.28	92.31	99.77	NA	74.19	NA	224
9	Hours	Template-based	Original	-	-	**46.38**	**42.11**	**35.34**	**80.16**	**63.92**	**100.00**	**100.00**	NA	NA	**100.00**	51
10	Hours	mT5	Original	B	15	35.82	35.16	25.90	61.18	40.91	15.38	99.50	NA	NA	83.17	137
11	Hours	mT5	Translated	B	30	40.22	37.92	29.01	60.06	41.30	92.31	**100.00**	NA	NA	77.51	**184**
12	Hours	mT5	Synthetic	B	50	42.62	39.83	31.69	63.38	45.28	**100.00**	99.83	NA	NA	98.06	163
13	Salary	Template-based	Original	-	-	**30.45**	**32.03**	**22.34**	**60.80**	**38.09**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	112
14	Salary	mT5	Original	A	30	28.27	31.16	20.53	46.82	28.10	96.88	99.85	89.71	99.06	85.31	528
15	Salary	mT5	Translated	A	40	29.20	31.67	20.56	48.22	28.57	96.88	99.93	99.21	97.44	84.23	585
16	Salary	mT5	Synthetic	A	40	27.70	29.74	18.70	46.32	25.89	**100.00**	99.93	99.93	97.44	68.06	**607**

Table 2: The results for each benefit section. NA refers to 'Not Applicable

4.3 Hours

For the experiments on the hours section, the best performing input format was B; *'FEATURE=value'*. The results show that by using the translated dataset, the model reaches a significantly higher language accuracy, with English language accuracy improving from 15.38% to 92.31% (row 11 on Table 2). However, by using the translated training set for fine-tuning, and having two samples for the same number, the model's input accuracy decreased to 77.51% (row 11). By using the synthetic training data for fine-tuning, thus adding more randomness, the input accuracy increased to 98.06% and approached the template-based method, while keeping a high language accuracy (row 12). Moreover, the experiments show that the transformers use about 3 times more words (137-184) than the template-based method (51).

4.4 Salary

For the salary section the best performing input format was input A. Row 13-16 on Table 2 illustrates that a high language accuracy (>96.88%) is achieved by the transformer regardless of the training data. While the categorical input generation accuracy was not significantly affected by the training data (decrease of 1.62%), the binary input accuracy improved (by 9.5%) with added translated data (row 15). In contrast, when fine-tuning on more training data, numeric input accuracy dropped, in particular with synthetic data (by 16.17%, row 16). Lastly, the vocabulary size is around 2.5 times higher for transformers (528-607) compared to the template-based method (127).

5 Analysis and discussion

5.1 Language accuracy

Table 2 shows how fine-tuning the model on different training data affected the language accuracy for each benefit section. The Dutch language accuracy was high (between 98.97% and 100%) for each benefit section regardless of the training data. In terms of English language accuracy, the location and salary sections performed well (with a language accuracy between 96.9% and 100%) regardless of the training data. While the extra training data was beneficial for the hours section (and lead to an accuracy of 92.3% from 15.4%), it was ineffective for the contract section, keeping the English language accuracy at zero. Meanwhile, by translating the input and the prefix, a good level of English language accuracy (92.31%) was obtained for the contract section too. While intuition suggests that language accuracy may be affected by the volume of training samples - as Dutch samples account for approximately 95% of each section - this idea cannot be supported, because both the contract and hours sections have approximately 3-4 times more training data (and approximately 2.5-3.5 times more English samples in the training data), than the location section, which was still able to reach a perfect English language accuracy with such a small training set as 454 samples (from which 20 were English).

5.2 Input generation accuracy

Appendix F contains three correct and three incorrect examples for each type of input generation (categorical, binary and numeric) and each benefit section.

Location The categorical input in the location benefit is unique because the input can only be generated in one single way (mentioning the city name can not be done differently). Row 2 on Table 2 shows that when fine-tuned on the original training data, the model achieved an input accuracy of 76.96%. Using the translated training data for fine-tuning, significantly increased this score and resulted in an accuracy of 84.87%. However, this is still a long way from the rule-based approach. Previous work (Kale and Rastogi, 2020) pointed out that data-to-text generation algorithms perform worse on test samples that have an unseen input. This decrease was major, around 23.2% (with an accuracy of 60%) for a neural-based model, but only 2% (with an accuracy of 90%) for T5 transformer, when applied in the closed Wikipedia domain (Kale and Rastogi, 2020). Our test set had five cities that were unseen; not present in the training data. The input accuracy for the 9 samples, which used one of these unseen cities, was 44.44%. Whereas the input accuracy for the samples with the seen input was 96.06%. In case of some samples with unseen inputs, the model generated text with a new city name. For example, when the input was *'Zwaag'* the model generated the following sentence *'Pal gelegen achter her centraal station Zwaaijdijk'*, naming a non-existing city. The samples with unseen inputs in our dataset are very specific, small Dutch city names, that were allegedly not part of the corpus mT5 was trained on, which could explain why transfer learning had a major impact when using the closed domain Wikipedia dataset (WebNLG) in the related work (Kale and Rastogi, 2020), but not in our domain.

Contract and salary We see a high accuracy for categorical input generation (96.44% and 99.06%) at the contract and salary sections, using only the original training data. Fine-tuning on extra training data had no significant impact on salary, however adding translated training data caused a drop in categorical input accuracy by 25.25% for the contract section.

Binary input generation The salary section of Table 2 reveals that fine-tuning the model on the translated dataset significantly improves binary input generation accuracy (from 89.71% to 99.21%). This accuracy nears perfection, which demonstrates the transformer can learn domain knowledge by learning to hide the salary amount. Some of the correctly generated text for hidden salaries were: *'Based on experience', 'A good salary and excellent reimbursement overtime'*.

Numeric input accuracy The hours section of Table 2 illustrate that fine-tuning on the translated training set resulted in some decrease (5.66%, from 83.17% to 77.51%) for the hours benefit section regarding the numeric input generation accuracy. This is not surprising given that by translating, all covered numbers in the training samples were covered roughly twice as much. However, by using the synthetic dataset for training the distribution of the samples in the training set was very even (because randomly choosing the values during the method of creating the synthetic data) and the model reached a 98.06% input generation accuracy. Compared to prior work (Gong et al., 2019) where including synthetic data lead to an accuracy increase of 2.6%, in our case accuracy increased by 14.89%. However, this strategy of fine-tuning on synthetic data did not work for the salary section. The accuracy of numeric input generation actually decreased significantly (from 85.31% to 68.06%). This might be explained by the nature of the data, because the hours section only includes integer numbers and covers a very limited range of numbers (between 2 and 55 in the training data). On the other hand the salary section contains monthly and hourly salaries too. Monthly salary are integers with a wide range (the range in the training data is: 250 - 6000) while the hourly wages are rounded to two decimals and cover the range (in the training data) between 9.63 - 45.00. A possible explanation might be that the original dataset focuses on certain numbers. The proportion of the unique numbers (for both minimum and maximum and hourly and monthly salaries) was between 20.5% and 36.05%. There are some patterns in the data, which were not matched when adding synthetic data. For example, next to understandably common decimal values for hourly salaries (0, 50, 99) other values (e.g., 48, 97, 29 or 8) are common too. This could be caused by collective labor agreements, minimum wage, yearly salary increases, or inflation correction.

5.3 Limitations

One limitation of our work is the unbalanced nature of the original dataset in terms of language. While this provides a unique opportunity and challenge, it also leads to having only very few (between 2-64) English samples for some sections. These small

sample sizes may make our findings less reliable regarding the English language accuracy. A second limitation concerns our focus on the benefit sections of vacancies. These sections are rather general and typically do not vary between jobs as much as other vacancy sections (e.g., the task section), which might make our finding less generalizable. However, due to absence of structured input we were unable to study other vacancy sections.

6 Conclusion

Although the amount and the imbalanced nature of the dataset (in terms of language) prevent us from drawing definite conclusions, our analysis reveals several insights into the behavior of the mT5 model, which we use to answer our three sub-questions.

RQ1.1 First, we focus on finding how accurately mT5 generates text in the correct language. Overall, we found that with the right parameter tuning, prefix, and input, mT5 performs very well in generating text in the correct language.

RQ1.2 To find how accurate mT5 is at including different types of inputs (numerical, binary, categorical) correctly in the generated text, we used our own new, domain-specific metrics in the generated text. We conclude that while mT5 is highly accurate at generating from binary inputs, and seen categorical inputs, it struggles with unseen categorical and numeric inputs.

RQ1.3 Finally, we explore how fine-tuning mT5 on translated and synthetic data affects the language and input generation accuracy. We found that fine-tuning mT5 on additional training data (translated and synthetic) can lead to major improvements, however the rate of which strongly depends on the nature of the task and the distribution of numeric samples in the dataset.

In conclusion, we evaluated how accurately mT5 includes inputs in the generated text with custom metrics developed by us, on a novel task in the job description domain. We applied transfer learning on a new and unique dataset in terms of volume and language balance. While our findings are mostly in line with earlier studies, we demonstrate that language accuracy, input generation accuracy, and the effectiveness of extra training data are highly dependent on the nature of data and task. Overall, even though the model may not be as accurate as a template-based approach, it can be employed under human supervision to generate benefit sections of vacancies, and will yield more diverse outputs. Additionally, when using the model, the users can rely on custom evaluation metrics developed by us.

Promising future work directions include using more languages and other sections of vacancy texts. For example, while the requirements section of the vacancies currently do not have structured input data available; these inputs may be extracted automatically using a skill extraction model, such as LinkedIn's Job2Skills model (Shi et al., 2020). Extending the study to novel sections of the vacancy could give a great opportunity to observe how transformers work with categorical input with a wider range of values than our `salarytype` and `contracttype`, while still being more general than the `location` input. Another intriguing step is to investigate how automatically generated text affects the accessibility of job descriptions. Finally, evaluating the models with the recruiters would be a logical next step too.

References

Oshin Agarwal, Mihir Kale, Heming Ge, Siamak Shakeri, and Rami Al-Rfou. 2020. Machine translation aided bilingual data-to-text generation and semantic parsing. In *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*, pages 125–130, Dublin, Ireland (Virtual). Association for Computational Linguistics.

Thiago Castro Ferreira, Claire Gardent, Nikolai Ilinykh, Chris van der Lee, Simon Mille, Diego Moussallem, and Anastasia Shimorina. 2020. The 2020 bilingual, bi-directional WebNLG+ shared task: Overview and evaluation results (WebNLG+ 2020). In *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*, pages 55–76, Dublin, Ireland (Virtual). Association for Computational Linguistics.

Jordan Clive, Kris Cao, and Marek Rei. 2021. Control prefixes for text generation. *arXiv preprint arXiv:2110.08329*.

Yuvaraja Devarajan. 2018. A study of robotic process automation use cases today for tomorrow's business. *International Journal of Computer Techniques*, 5(6):12–18.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Bhuwan Dhingra, Manaal Faruqui, Ankur Parikh, Ming-Wei Chang, Dipanjan Das, and William Cohen. 2019. Handling divergent reference texts when evaluating table-to-text generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4884–4895, Florence, Italy. Association for Computational Linguistics.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. Creating training corpora for nlg micro-planning. In *55th annual meeting of the Association for Computational Linguistics (ACL)*.

Li Gong, Josep Crego, and Jean Senellart. 2019. Enhanced transformer model for data-to-text generation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 148–156, Hong Kong. Association for Computational Linguistics.

Mihir Kale and Abhinav Rastogi. 2020. Text-to-text pre-training for data-to-text tasks. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 97–102, Dublin, Ireland. Association for Computational Linguistics.

Karen Kukich. 1983. Design of a knowledge-based report generator. In *Proceedings of the 21st Annual Meeting on Association for Computational Linguistics*, ACL '83, page 145–150, USA. Association for Computational Linguistics.

Alon Lavie and Abhaya Agarwal. 2007. Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments. In *Proceedings of the second workshop on statistical machine translation*, pages 228–231.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Kathleen McKeown. 1992. *Text generation*. Cambridge University Press.

Susan W McRoy, Songsak Channarukul, and Syed S Ali. 2000. Yag: A template-based generator for real-time systems. In *INLG'2000 Proceedings of the First International Conference on Natural Language Generation*, pages 264–267.

Diego Moussallem, Dwaraknath Gnaneshwar, Thiago Castro Ferreira, and Axel-Cyrille Ngonga Ngomo. 2020. Nabu – multilingual graph-based neural rdf verbalizer. In *The Semantic Web – ISWC 2020*, pages 420–437, Cham. Springer International Publishing.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Ankur Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020. ToTTo: A controlled table-to-text generation dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1173–1186, Online. Association for Computational Linguistics.

Chuan Qin, Kaichun Yao, Hengshu Zhu, Tong Xu, Dazhong Shen, Enhong Chen, and Hui Xiong. 2022. Towards automatic job description generation with capability-aware neural networks. *IEEE Transactions on Knowledge and Data Engineering*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR.

Baoxu Shi, Jaewon Yang, Feng Guo, and Qi He. 2020. Salience and market-aware skill extraction for job targeting. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2871–2879.

Lu Wang and Claire Cardie. 2013. Domain-independent abstract generation for focused meeting summarization. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1395–1405.

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2017. Challenges in data-to-document generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2253–2263, Copenhagen, Denmark. Association for Computational Linguistics.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online. Association for Computational Linguistics.

Giulio Zhou and Gerasimos Lampouras. 2020. WebNLG challenge 2020: Language agnostic delexicalisation for multilingual RDF-to-text generation. In *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*, pages 186–191, Dublin, Ireland (Virtual). Association for Computational Linguistics.

Appendix A Template-based method

This appendix helps to understand the template-based method described in subsection 3.2. It contains the pipeline for the template-based method with an example (Figure 2) as well as the combinations for the templates.

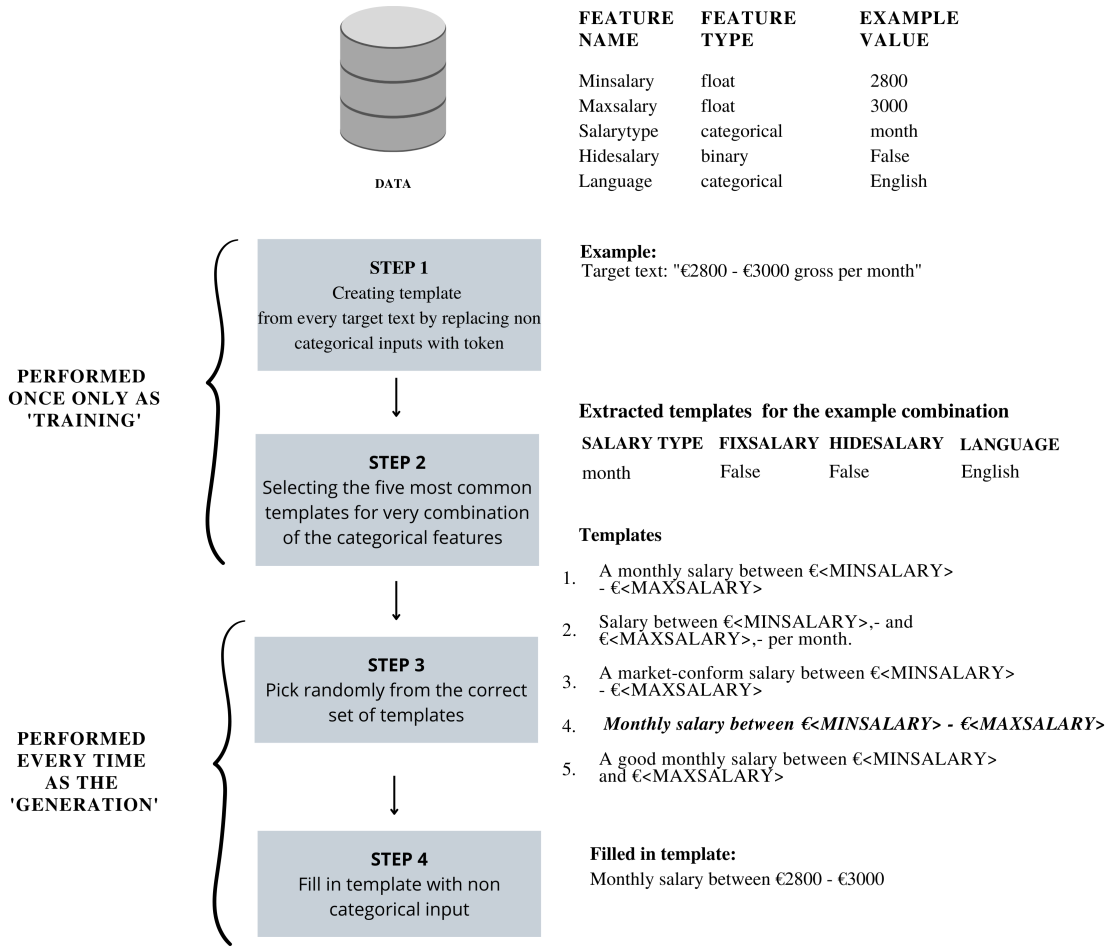

Figure 2: The pipeline of the template-based method with an example for the salary benefit section. In the first step the values of all non categorical features get replaced by the feature name resulting in a template for every training sample. Next, for all combination the five most common template gets extracted. In the third step, one of the chosen template gets randomly picked which then is filled in by replacing the non categorical feature names with their values in step 4.

A.1 Template combinations

This Section contains the combination of templates for each benefit section. Table 3 shows that the location section only had two different types of templates, the contract and the hours benefit sections had six (show on Table 4 and Table 5), while the salary benefit section had ten different types (shown on Table 6).

Combination	Language
1	English
2	Dutch

Table 3: There are 2 combinations for the location section, each combination contains 5 templates.

Combination	Language	Fix hour	Full time
1.	English	False	False
2.	English	True	False
3.	English	True	True
4.	Dutch	False	False
5.	Dutch	True	False
6.	Dutch	True	True

Table 4: There are 6 combinations for the hours section, each combination contains 5 templates.

Combination	**Language**	**Contract type**
1	English	Temporary
2	English	Temporary with a possibility to fixed
3	English	Fixed
4	Dutch	Temporary
5	Dutch	Temporary with a possibility to fixed
6	Dutch	Fixed

Table 5: There are 6 combinations for the contract section, each combination contains 5 templates.

Combination	**Language**	**Hidesalary**	**Fixsalary**	**Salary type**
1.	English	True	-	-
2.	Dutch	True	-	-
3.	English	False	True	Hourly
4.	English	False	True	Monthly
5.	Dutch	False	True	Hourly
6.	Dutch	False	True	Monthly
7.	English	False	False	Hourly
8.	English	False	False	Monthly
9.	Dutch	False	False	Hourly
10.	Dutch	False	False	Monthly

Table 6: There are 10 combinations for the salary section, each combination contains 5 templates.

Appendix B mT5 model

For optimizer we used the Adafactor optimizer, with the default parameters suggested by Shazeer and Stern (2018). Table 7 shows the values of the parameters.

Parameter	**Parameter name**	**Set value**
ϵ_1	regularization constants for square gradient	10^{-30}
ϵ_2	parameter scale	10^{-3}
d	threshold of root mean square of final gradient update	1
$\hat{\beta}_{2t}$	decay rate	$1 - t^{-0.8}$
α	external learning rate	10^{-3}

Table 7: The hyperparameter settings for the Adafactor optimezer based on Shazeer and Stern (2018)

Appendix C Details of dataset, pre- and postprocessing steps

First, we fixed the monetary units and the `hidesalary` binary field for the salary benefit section. Then, we excluded samples that were not correct based on the language and at including the input (for which we used our own input accuracy metrics explained in subsection 3.6).
In Table 8, we show the set of words for each benefit section that were removed before running the language detection on the generated text.

Section	**Words removed for language detection**
Location	`location` from input
Contract	words : contract, direct, per, week, of, permanent
Hours	-
Salary	per, week

Table 8: Words removed for each section before running the language detection on the generated text

In Table 9, we show the size of training, validation and test set for each benefit sections.

Benefits section	**Size of training set**	**Size of validaiton set**	**Size of test set**
Location	454	152	152
Contract	2696	899	899
Hours	1854	618	618
Salary	4197	1399	1399

Table 9: Size of training, validation and test set for each benefit section

In Table 10, we show the number of English and Dutch samples for each benefit section in the training, validation and test set.

Section	**Samples in training set**			**Samples in validation set**			**Samples in test set**		
	All	**Dutch**	**English**	**All**	**Dutch**	**English**	**All**	**Dutc**	**English**
Location	454	434	20	152	148	4	152	150	2
Contract	2696	2626	70	899	881	18	899	873	26
Hours	1854	1807	47	618	608	10	618	605	13
Salary	4197	4061	136	1399	1353	46	1399	1335	64

Table 10: The number of samples for each section in the training, validation and test set. The sample size is further braked down regarding the language.

Appendix D Random seed

The random seed was set to 1 for the splitting the dataset into training, validation and test set. The validation set was used for hyperparameter tuning and deciding which input format to use. Additionally, because both the template-based method and the transformer method has some randomness we assigned a random integer number between 1 and 9999 to every sample in the validation and the test set, which we then used as the random seed for generating the text, for the template-based method and the mT5 models. This way we ensured reproducibility and fair comparison between models. Additionally, we set the random seed for the transformer models to be 1 before fine-tuning the models.

Appendix E Rule-based classifier for the contract type evaluation

The following algorithm contains the rule-based classifier that was used to detect the contract type in the generated text for the contract benefit section.

Algorithm 1 Contract type classifier

```
words = set of lower case words of generated text
if any words in {'prospect', 'possibility' , 'opportunity' , 'chance', 'kans', 'mogelijkheid' , 'uitzicht' ,
'view', 'option'} then
    return Temporary with a possibility to fixed
else if any words in { 'temporary', 'tijdelijk' } then
    return Temporary
else if any words in {'permanent', 'direct', 'directly', 'long', 'vast', 'dienstverband', 'jaarcontract'}
then
    return Fixed
end if
```

Appendix F Examples

The following tables contain 3 correct and 3 incorrect examples for each type of input (categorical, binary, numeric) for each section. The Dutch samples have been translated by the authors to English for a better understanding of the examples.

F.1 Examples of categorical input generation

Examples of correct samples from test set		**Examples of incorrect samples from test set**	
Input	**Output**	**Input**	**Output**
LOCATION = Veghel	Super mooi bekend bedrijf in Veghel (Translated: Super nice well-known company in Veghel)	LOCATION = RIJEN	Werken in de omgeving van Rijk (Translated: Work in the area of Rijk)
LOCATION = Schiphol	Werken in de dynamische omgeving Schiphol (Translated: Work in the dynamic environment of Schiphol)	LOCATION = Zwaag	Pal gelegen achter het centraal station Zwaaijdijk! (Translated: Located right behind Zwaaijdijk central station!)
LOCATION = Amstelveen	Werken op een mooie locatie in Amstelveen (Translated: Work at a beautiful location in Amstelveen)	LOCATION = Oud Gastel	Een baan in de Hoeksche Waard: Oud-Beijerland (Translated: A job in the Hoeksche Waard: Oud-Beijerland)

Table 11: Correct and incorrect examples of the categorical location generation from the test set

Examples of correct samples from test set		**Examples of incorrect samples from test set**	
Input	**Output**	**Input**	**Output**
<CONTRACTTYPE>temporary with a view to permanent	The chance of permanent contract	<CONTRACTTYPE>temporary with a view to permanent	Fulltime with our client
<CONTRACTTYPE>temporary with a view to permanent	Opportunity to get a permanent contract	<CONTRACTTYPE>fix	Een temporary baan (Translated: A temporary job)
<CONTRACTTYPE>vast	Direct op contract bij het bedrijf (Translated: Direct contract with the company)	<CONTRACTTYPE>fix	The possibility to get a permanent contract

Table 12: Correct and incorrect examples of the categorical contract generation from the test set

Examples of correct samples from test set		**Examples of incorrect samples from test set**	
Input	**Output**	**Input**	**Output**
<FIXSALARY>2100.0 <SALARYTYPE>per month	A salary of € 2100 gross per month.	<FIXSALARY>10.0 <SALARYTYPE>per hour	Salaris €10. (Translated: Salary €10.)
<MINSALARY>2600.0 <MAXSALARY>3000.0 <SALARYTYPE>per month	Salary between € 2600 and € 3000 gross per month.	<MINSALARY>11.0 <MAXSALARY>12.0 <SALARYTYPE>per hour	Salaris tussen de €11,- en €12 (Translated: Salary between €11 and €12.)
<FIXSALARY>14.0 <SALARYTYPE>per hour	Good hourly rate of € 14 gross per hour.	<FIXSALARY>11.5 <SALARYTYPE>per hour	Salaris Bespreekbaar, per week betaald! (Translated: Salary Negotiable, paid per week!)

Table 13: Correct and incorrect examples of the categorical salary generation from the test set

F.2 Examples of binary input generation

Examples of correct samples from test set		Examples of incorrect samples from test set	
Input	**Output**	**Input**	**Output**
<HIDESALARY>TRUE	Based on experience	<HIDESALARY>TRUE	Een salaris van €11,- bruto per uur! (Translated: A salary of €11 gross per hour!)
<HIDESALARY>TRUE	A good salary and excellent reimbursement overtime	<HIDESALARY>TRUE	Een salaris van €11,49 per uur (Translated: A salary of €11.49 per hour)
<HIDESALARY>TRUE	A good salary in accordance with the collective labor agreement	<HIDESALARY>TRUE	Een loon van € 16,- per uur (Translated: A wage of €16 per hour)

Table 14: Correct and incorrect examples of the binary salary generation from the test set

F.3 Examples of numeric input generation

Examples of correct samples from test set		Examples of incorrect samples from test set	
input	**output**	**inpug**	**output**
FIXHOUR = 32	Work week of 32 hours	FIXHOUR = 32	De mogelijkheid om 32 ur tot 32 uur te werken (Translated: Possibility to work from 32 to 32 hours)
MINHOUR = 24| MAXHOUR = 40	24 to 40 hours, choose how many hours you work	MINHOUR = 10| MAXHOUR = 15	11-15 uur, vraag naar het rooster. (Translated: 11-15 hours, ask for the schedule.
MINHOUR = 32| MAXHOUR = 40	A working week from 32 to 40 hours (your preference)	FIXHOUR = 38	Een werkweek tussen de 38 en 38 uur. (Translated: A work week between 38 and 38 hours.)

Table 15: Correct and incorrect examples of the numeric hours generation from the test set

	Examples of correct samples from test set		Examples of incorrect samples from test set	
	Input	**Output**	**Input**	**Output**
Montlhy	<FIXSALARY>500.0 <SALARYTYPE>per month	max. 500 euro gross per month	<MINSALARY>2400.0 <MAXSALARY>3100.0 <SALARYTYPE>per month	Salaris tussen de €2400 en €.- €3300 per maand (Translated: Salary between €2400 and €.- €3300 per month)
	<FIXSALARY>500.0 <SALARYTYPE>per month	Stage starting salary of €500 per month	<FIXSALARY>2200.0 <SALARYTYPE>per month	Goed salaris van rond de €2200 en €2200 per maand! (Translated: Good salary of around €2200 and €2200 per month!)
	MINSALARY>3000.0 <MAXSALARY>3500.0 <SALARYTYPE>per month	Salary between €3000 and €3500 per month	<MINSALARY>2771.0 <MAXSALARY>3934.0 <SALARYTYPE>per month	Salaris tussen de €2110 en €3738 bruto per maand (Translated: Salary between €2110 and €3738 gross per month)
Hourly	FIXSALARY>11.0 <SALARYTYPE>per hour	11 euros per hour and shifts allowances	<FIXSALARY>11.67 <SALARYTYPE>per hour	Een salaris van €11,62 bruto per uur (Translated: A salary of €11.62 gross per hour)
	<MINSALARY>14.27 <MAXSALARY>17.68 <SALARYTYPE>per hour	Salaris tussen de €14,27 en €17,68 per uur (Translated: Salary between €14.27 and €17.68 per hour)	<MINSALARY>12.42 <MAXSALARY>18.47 <SALARYTYPE>per hour	€12,42 - €18,57 per uur op basis van ervaring (Translated: €12.42 - €18.57 per hour based on experience)
	<FIXSALARY>11.5 <SALARYTYPE>per hour	Een lekker salaris van €11,50 per uur (Translated: A nice salary of €11.50 per hour)	<FIXSALARY>11.27 <SALARYTYPE>per hour	Uurloon van €12,27 (Translated: Hourly wage of €12.27)

Table 16: Correct and incorrect examples of the numeric salary generation from the test set