Attention Is Indeed All You Need: Semantically Attention-Guided Decoding for Data-to-Text NLG

Juraj Juraska and **Marilyn Walker**
Natural Language and Dialogue Systems Lab
University of California, Santa Cruz
{jjuraska,mawalker}@ucsc.edu

Abstract

Ever since neural models were adopted in data-to-text language generation, they have invariably been reliant on extrinsic components to improve their semantic accuracy, because the models normally do not exhibit the ability to generate text that reliably mentions all of the information provided in the input. In this paper, we propose a novel decoding method that extracts interpretable information from encoder-decoder models' cross-attention, and uses it to infer which attributes are mentioned in the generated text, which is subsequently used to rescore beam hypotheses. Using this decoding method with T5 and BART, we show on three datasets its ability to dramatically reduce semantic errors in the generated outputs, while maintaining their state-of-the-art quality.

1 Introduction

Task-oriented dialogue systems require high semantic fidelity of the generated responses in order to correctly track what information has been exchanged with the user. Therefore, their natural language generation (NLG) components are typically conditioned on structured input data, performing *data-to-text* generation. To achieve high semantic accuracy, neural models for data-to-text NLG have invariably been reliant on extrinsic components or methods. While large pretrained generative language models (LMs), such as GPT-2 or T5, perform better in this respect, even they do not normally generate text that reliably mentions all the information provided in the input.

In this work, we study the behavior of attention in large pretrained LMs fine-tuned for data-to-text NLG tasks. We show that *encoder-decoder* models equipped with *cross-attention* (i.e., an attention mechanism in the decoder looking back at the encoder's outputs) are, in fact, aware of the semantic constraints, yet standard decoding methods do not fully utilize the model's knowledge. The method we propose extracts interpretable information from the model's cross-attention mechanism at each decoding step, and uses it to infer which slots have been correctly realized in the output. Coupled with beam search, we use the inferred slot realizations to rescore the beam hypotheses, preferring those with the fewest missing or incorrect slot mentions.

To summarize our contributions, the proposed semantic attention-guided decoding method, or SEA-GUIDE for short: **(1)** drastically reduces semantic errors in the generated text (shown on the E2E, ViGGO, and MultiWOZ datasets); **(2)** is domain- and model-independent for encoder-decoder architectures with cross-attention, as shown on different sizes of T5 and BART; **(3)** works out of the box, but is parameterizable, which allows for further optimization; **(4)** adds only a small performance overhead over beam search decoding; and **(5)** perhaps most importantly, requires no model modifications, no additional training data or data preprocessing (such as augmentation, segmentation, denoising, or alignment), and no manual annotation.[1]

2 Related Work

Several different approaches to enhancing semantic accuracy of neural end-to-end models have been proposed for data-to-text NLG over the years. The most common approach to ensuring semantic quality relies on over-generating and then reranking candidate outputs using criteria that the model was not explicitly optimized for in training. Reranking in sequence-to-sequence models is typically performed by creating an extensive set of rules, or by training a supplemental classifier, that indicates for each input slot whether it is present in the output utterance (Wen et al., 2015a; Dušek and Jurčíček,

[1]The code for SEA-GUIDE and heuristic semantic error evaluation can be found at https://github.com/jjuraska/data2text-nlg.

Proceedings of the 14th International Conference on Natural Language Generation (INLG), pages 416–431,
Aberdeen, Scotland, UK, 20-24 September 2021.

2016; Juraska et al., 2018; Agarwal et al., 2018; Kedzie and McKeown, 2020; Harkous et al., 2020).

Wen et al. (2015b) proposed an extension of the underlying LSTM cells of their sequence-to-sequence model to explicitly track, at each decoding step, the information mentioned so far. The coverage mechanism (Tu et al., 2016; Mi et al., 2016; See et al., 2017) penalizes the model for attending to the same parts of the input based on the cumulative attention distribution in the decoder. Chisholm et al. (2017) and Shen et al. (2019) both introduce different sequence-to-sequence model architectures that jointly learn to generate text and reconstruct the input facts. An iterative self-training process using data augmentation (Nie et al., 2019; Kedzie and McKeown, 2019) was shown to reduce semantic NLG errors on the E2E dataset (Novikova et al., 2017). Among the most recent efforts, the jointly-learned segmentation and alignment method of Shen et al. (2020) improves semantic accuracy while simultaneously increasing output diversity. Kedzie and McKeown (2020) use segmentation for data augmentation and automatic utterance planning, which leads to a reduction in semantic errors on both the E2E and ViGGO (Juraska et al., 2019) datasets.

In contrast to the above methods, our approach does not rely on model modifications, data augmentation, or manual annotation. Our method is novel in that it utilizes information that is already present in the model itself to perform semantic reranking.

Finally, related to our work is also controllable neural language generation, in which the constrained decoding strategy is often used, rescoring tokens at each decoding step based on a set of feature discriminators (Ghazvininejad et al., 2017; Baheti et al., 2018; Holtzman et al., 2018). Nevertheless, this method is typically used with unconditional generative LMs, and hence does not involve input-dependent constraints.

3 Semantic Attention-Guided Decoding

While we will evaluate the SEA-GUIDE method on ViGGO, E2E, and MultiWOZ, we develop the method by careful analysis of the cross-attention behavior of different pretrained generative LMs fine-tuned on the ViGGO dataset. ViGGO is a parallel corpus of structured meaning representations (MRs) and corresponding natural-language utterances in the video game domain. The MRs consist of a dialogue act (DA) and a list of slot-and-value pairs. The motivation for selecting ViGGO for developing the method was that it is the smallest dataset, but it provides a variety of DA and slot types (as shown in Table 1). The models used for the analysis were the smallest variants of T5 (Raffel et al., 2020) and BART (Lewis et al., 2020). We saved the larger variants of the models, as well as the other two datasets, for the evaluation.

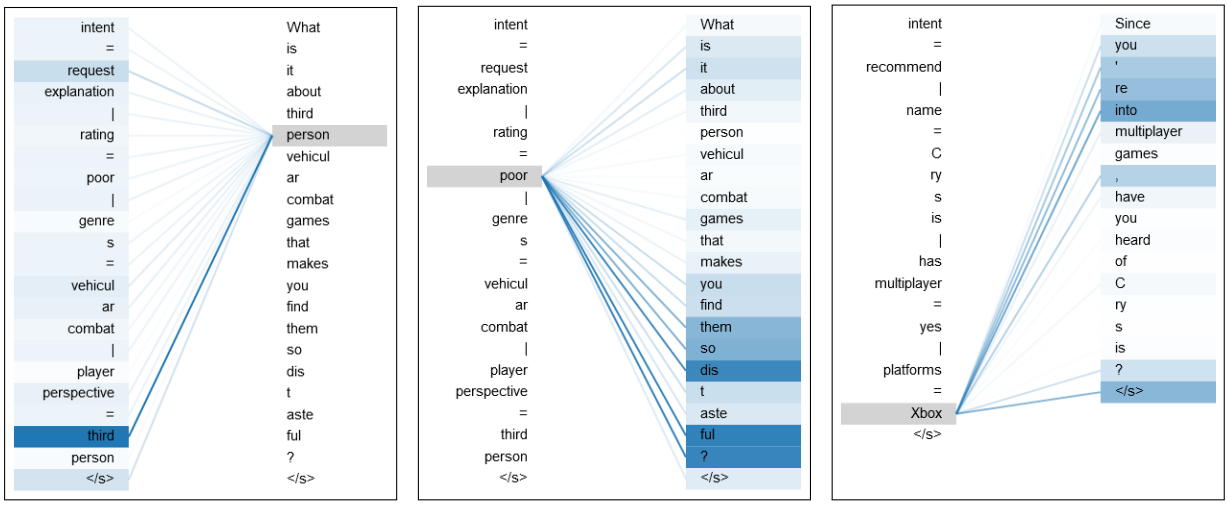
3.1 Interpreting Cross-Attention

Attention (Bahdanau et al., 2015; Luong et al., 2015) is a mechanism that was introduced in encoder-decoder models (Sutskever et al., 2014; Cho et al., 2014) to overcome the long-range dependencies problem of RNN-based models. It allows the decoder to effectively condition its output tokens on relevant parts of the encoder's output at each decoding step. The term *cross-attention* is primarily used when referring to the more recent transformer-based encoder-decoder models (Vaswani et al., 2017), to distinguish it from the *self-attention* layers present in both the encoder and the decoder transformer blocks. The cross-attention layer ultimately provides the decoder with a weight distribution at each step, indicating the importance of each input token in the current context.

Our results below will show that visualizing the attention weight distribution for individual cross-attention layers in the decoder – for many different inputs – reveals multiple universal patterns, whose combination can be exploited to track the presence, or lack thereof, of input slots in the output sequence. Despite the differences in the training objectives of T5 and BART, as well as their different sizes, we observe remarkably similar patterns in their respective cross-attention behavior. Below, we describe the three most essential patterns (illustrated in Figure 1) that we use in SEA-GUIDE.

3.1.1 Verbatim Slot Mention Pattern

The first pattern consistently occurs in the lowest attention layer, whose primary role appears to be to retrospectively keep track of a token in the input sequence that the decoder just generated in the previous step. Figure 1a shows an example of an extremely high attention weight on the input token "third" when the decoder is deciding which token to generate after "What is it about *third*" (which ends up being the token "person"). This pattern, which we refer to as the *verbatim* slot mention pattern, can be captured by maximizing the weight over all attention heads in the decoder's first layer.



(a) Verbatim slot mention (1st layer). (b) Paraphrased slot mention (3rd layer). (c) Unrealized slot mention (4th layer).

Figure 1: Visualization of cross-attention weight distribution for the 6-layer T5-small (trained on the ViGGO dataset) in 3 different scenarios. The left column in each corresponds to the input tokens, and the right to the tokens generated by the decoder. The darker the blue background shade, the greater the attention weight. Note that the weights are aggregated across all attention heads by extracting the maximum.

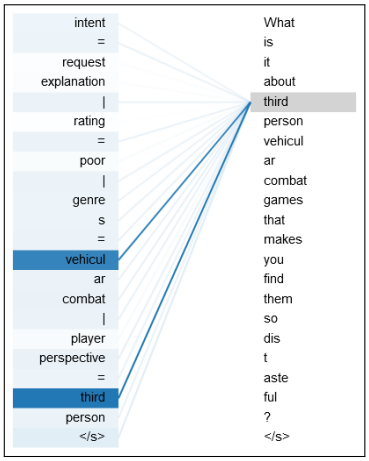


Figure 2: An example of the decoder paying equal attention (in the 5th layer of the 6-layer T5-small) to two slots in the input sequence when deciding what to generate next after "What is it about".

3.1.2 Paraphrased Slot Mention Pattern

Paraphrased slot mentions, on the other hand, are captured by the higher layers, at the moment when a corresponding token is about to be mentioned next. Essentially, as we move further up the layers, the cross-attention weights gradually shift towards input tokens that correspond to information that is most likely to *follow next* in the output, and capture increasingly more abstract concepts in general. Figure 1b shows an example of the RATING slot's value "poor" paraphrased in the generated utterance as "distasteful"; the first high attention value associated with the input token "poor" occurs when the decoder is about to generate the "dis" token.

At certain points during generation, however, the attention in the uppermost layers is distributed fairly evenly among multiple slots, because any of them could lead to a coherent continuation of the sentence. For example, the generated utterance in Figure 2 could have started with "What is it about vehicular combat games played from a third-person perspective that…", where the GENRES slot is output before the PLAYER PERSPECTIVE slot.

In order to recognize a paraphrased mention, without incorrectly capturing other slots considered, we propose averaging the cross-attention weights, using only the bottom half of the layers (e.g., layers 1 to 3 in the T5-small model).

3.1.3 Unrealized Slot Mention Pattern

The third pattern alleviates any undesired side effects of identifying paraphrased mentions using the second pattern, i.e., slots incorrectly assumed to be mentioned. Figure 1c illustrates an unrealized slot (PLATFORMS) being paid attention to in several decoding steps. The cross-attention weight distribution for the "Xbox" token in the 4th layer, shows that the decoder considered mentioning the slot at step 5 (e.g., "Since you're *an Xbox fan* and like multiplayer games,…"), as well as step 8 (e.g., "…into multiplayer games *on Xbox,*…"). The second pattern, depending on the sensitivity setting (see Section 3.2), might infer the PLATFORMS slot as a paraphrased mention at step 5 and/or 8.

However, the PLATFORMS slot's value is also paid attention to when the decoder is about to generate the EOS token and, importantly, without any

high attention weights associated with other slots at this step. This suggests that the model *is aware* that it omitted that slot. However, at that point, the decoder is more confident ending the sentence than realizing the missed slot after generating a question mark. This *unrealized* slot mention pattern is most likely to occur in the higher cross-attention layers, but not necessarily, so it is more effective to capture it by averaging the attention weights over all layers (at the last decoding step).

Note on Boolean Slots. With any of the three patterns described above, Boolean slots, such as HAS MULTIPLAYER in Figure 1c, typically have a high attention weight associated with their name rather than the value. This observation leads to a different treatment of Boolean slots, as described in Appendix B.1.

3.2 Slot Mention Tracking

We use the findings of the cross-attention analysis for automatic slot mention tracking in the decoder. During decoding, for each sequence, the attention weights associated with the next token to be generated are aggregated as per Section 3.1. Using configurable *thresholds*, the aggregated weights are then binarized, i.e., set to 1 if above the threshold, and 0 otherwise. This determines the sensitivity of the pattern recognition. Optionally, all but the maximum weight can be set to 0, in which case only a single input token will by implied even if the attention mass is spread evenly across multiple tokens. Finally, the indices of binarized weights of value 1, if any, are matched with their corresponding slots depending on which slot-span in the input sequence they fall into. For details on automatically extracting slot spans, see Appendix B.1.

3.2.1 Mention-Tracking Components

The three mention-tracking components, each of which operates on different attention layers and uses a different weight aggregation and binarization strategy, are summarized in Table 3. These components are executed in sequence and update one common slot-tracking object.

The first component, which tracks verbatim mentions, operates on the first attention layer only, with a high binarization threshold. Slot mentions identified by this component are regarded as high-confidence. The second component tracks paraphrased mentions, which are identified as slot mentions with low confidence, due to the partial ambiguity in mention detection using the second pattern (see Section 3.1.2). The third component only kicks in when the EOS token is the most probable next token. At that point, it identifies – with high sensitivity – slots that were not realized in the sequence (e.g., the PLATFORMS slot in Figure 1c), and removes the corresponding mention record(s). Only low-confidence mentions can be erased, while high-confidence ones are final once they are detected.

3.3 Semantic Reranking

Combining the slot mention tracking with beam search, for each input MR we obtain a pool of candidate utterances along with the semantic errors inferred at decoding time. We then rerank the candidates and pick the one with the fewest errors, resolving ties using the length-weighted log-probability scores determined during beam search.

4 Evaluation

In order to measure the proposed decoding method's performance in semantic error reduction, we first develop an *automatic* way of identifying erroneous slot mentions in generated utterances. In a human evaluation we establish that its performance is nearly perfect for all three datasets used for testing our models (see Section 4.1). We then use it to calculate the slot error rate (SER) automatically for all our model outputs across all datasets and configurations tested, which would be infeasible to have human annotators do.

Datasets. Besides ViGGO, which we use for fine-tuning the decoding (slot-tracking) parameters of the proposed SEA-GUIDE method, we evaluate its effectiveness for semantic error reduction on two *unseen* and *out-of-domain* datasets. While E2E (Novikova et al., 2017) is also a simple MR-to-text generation dataset (in the restaurant domain), MultiWOZ 2.1 (Eric et al., 2020) is a dialogic corpus covering several domains from which we extract system turns only, along with their MR annotations, along the lines of Peng et al. (2020) and Kale and Rastogi (2020). Table 1 gives an overview of the datasets' properties.

Setup. In our experiments, we fine-tune T5 and BART models of varying sizes on the above datasets' training partitions, select the best model checkpoints based on the BLEU score they achieve on the respective validation set, and evaluate them on the test sets while using different decoding meth-

	Size	Domains	DAs	Slots
ViGGO	6,900	1	9	14
E2E	51,426	1	1	8
MultiWOZ	70,530	7	13	27

Table 1: Dataset statistics, including the total number of dialogue act (DA) and slot types. For MultiWOZ, the numbers are calculated across system turns only.

	SER_{SA}	SER CI (95%)	Precision	IAA
ViGGO	2.77%	$2.19 \pm 1.55\%$	97.37%	1.00
E2E	3.98%	$3.91 \pm 1.73\%$	100%	1.00
MultiWOZ	1.19%	$1.35 \pm 0.91\%$	94.89%	0.90

Table 2: Human evaluation of the slot aligner's performance on each dataset. The IAA column indicates the Krippendorff's alpha reliability coefficient.

ods for inference. For beam search decoding, including when used as part of SEA-GUIDE, we use beam size 10 and early stopping, unless stated otherwise. All of our results are averaged over three runs with random initialization. For further details on training and inference parameters, we refer the reader to Appendix A.3.

4.1 Automatic Slot Error Evaluation

We evaluate our trained models performance with the standard NLG metrics BLEU (Papineni et al., 2002), METEOR (Lavie and Agarwal, 2007), ROUGE-L (Lin, 2004), and CIDEr (Vedantam et al., 2015), whose calculation is detailed in Appendix A.4. However, we also put substantial effort into developing a highly accurate heuristic *slot aligner* to calculate the semantic accuracy of generated utterances. The slot aligner is rule-based and took dozens of man-hours to develop, but it is robust and extensible to new domains, so it works on all three test datasets. Using the slot aligner, we count missed, incorrect, and repeated slot mentions, and determine the slot error rate (SER) as the percentage of these errors out of all slots.

To verify our slot aligner's performance, we take the generated utterances of one model per dataset for which it calculated a relatively high SER (indicated in the SER_{SA} column in Table 2). We then have one of the authors and an additional expert annotator manually label all of the errors as true or false positives. This corresponds to 38, 173 and 176 errors for ViGGO, E2E and MultiWOZ, respectively. From that we calculate the precision for each dataset, which turns out to be above 94% for each of the datasets. The almost perfect inter-annotator agreement (IAA), besides validating the precision, also suggests that the SER is an objective metric, and therefore well-suited for automation.

	Verbatim	Paraphrased	Unrealized
Layer agg.	1^{st} layer only	avg. over bottom half of layers	avg.
Head agg.	max.	max.	max.
Bin. threshold	0.9	0.4 (T5-small) 0.3 (BART-base)	0.1
Bin. max.	yes	no	no

Table 3: The final configuration of parameters used in each of the 3 mention-tracking components. The "Bin. max." row indicates whether only the maximum weight is kept during binarization, or all above the threshold.

Furthermore, we take samples of 72 ($\approx$ 20%), 63 ($\approx$ 10%) and 290 ($\approx$ 4%) of the generated utterances on ViGGO, E2E and MultiWOZ, respectively, annotate them for all types of errors, and calculate the *actual* SER confidence intervals (middle column). Their good alignment with the slot aligner SER scores, together with the high error classification precision, leads us to the conclusion that the slot aligner performs similarly to humans in identifying semantic errors on the above datasets.

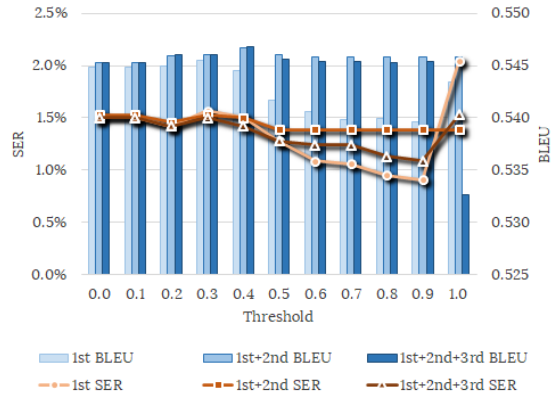
Besides SER evaluation, the slot aligner can also be used for beam reranking. Due to the handcrafted and domain-specific nature of the slot aligner, beam search with this reranking has a distinct advantage over SEA-GUIDE, which can be used for any domain out of the box. We therefore consider the results when using the slot-aligner reranking to be an upper bound for SEA-GUIDE in terms of SER.

4.2 SEA-GUIDE Parameter Tuning

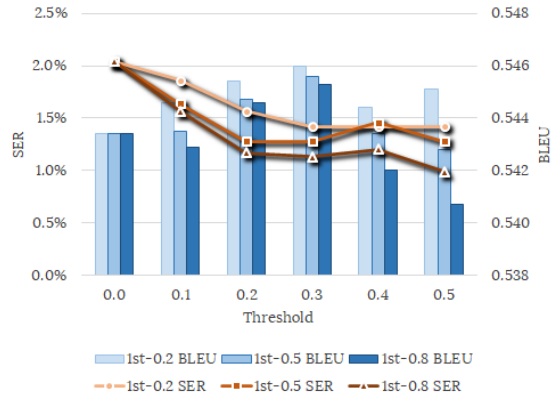
Each of the three mention-tracking components described in Section 3.2.1 has four configurable parameters, which we tuned by testing T5-small and BART-base, fine-tuned on the ViGGO dataset and equipped with SEA-GUIDE for inference. The parameter optimization was based on the insights obtained in Section 3.1 and a subsequent grid search, with results in Table 3.

For attention weight aggregation, we experimented with summing, averaging, maximizing, and normalizing. We determined *averaging* over layers and *maximizing* over heads to be the best combination for all three components. As for the binarization thresholds, Figure 3 shows the most relevant slice of the grid search space for each component, leading to the final threshold values.

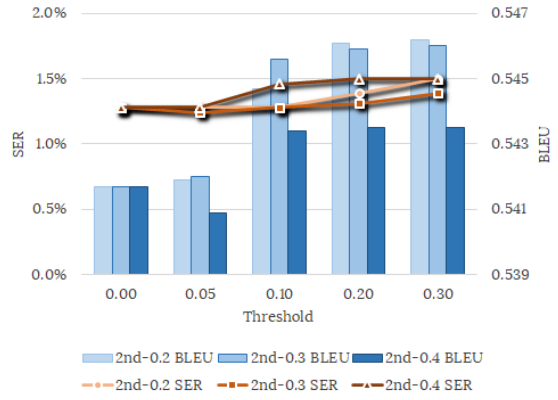
To show the effect of each slot-tracking component, we perform an ablation study with individual



(a) Threshold optimization for the 1st component (verbatim mentions), with the other components enabled or disabled. When enabled, the 2nd component's threshold was fixed at 0.3, and that of the 3rd at 0.1. Note that the threshold of 1.0 is equivalent to the 1st component being disabled, as attention weights are in the [0.0, 1.0] range.



(b) Threshold optimization for the 2nd component (paraphrased mentions), with the 1st component's threshold of 0.2, 0.5 and 0.8, and that of the 3rd component fixed at 0.1.



(c) Threshold optimization for the 3rd component (unrealized mentions), with the 2nd component's threshold of 0.2, 0.3 and 0.4, and that of the 1st component fixed at 0.5.

Figure 3: Effects of different parameter configurations of the 3 mention-tracking components on SER and BLEU of utterances generated by BART-base fine-tuned on ViGGO.

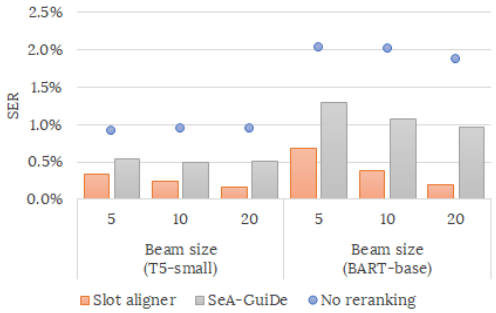


Figure 4: The effect of different beam size on the SER using different reranking methods on the ViGGO dataset. With greedy search decoding, the SER is 1.65% and 2.70% for T5 and BART, respectively.

components disabled.[2] As the plot in Figure 3a demonstrates, the 1st component by itself reduces the SER the most, but at the expense of the BLEU score, which decreases as the SER does – to the point where BLEU drops below 0.54 when the SER is at its lowest (0.91%), that is with a threshold of 0.9. For reference, the SER and the BLEU score achieved with beam search only are 2.04% and 0.543, respectively. Adding the 2nd component brings the BLEU score up to above 0.545, nevertheless the SER jumps to 1.39%. Finally, enabling the 3rd component too has a negligible negative effect on BLEU, but reduces the SER to 1.09%.

Figure 3b shows that the 2nd component gives optimal performance when its threshold is set to around 0.3. This setting maximizes BLEU, while keeping SER low. Beyond 0.3 the BLEU score starts dropping fast, and with a threshold of greater than 0.5, the 2nd component has barely any effect anymore. Similarly, Figure 3c shows the threshold value of 0.1 to be optimal in the 3rd component, when optimizing for both metrics. Thresholds higher than 0.3 cut off almost all aggregated weights in this component, virtually disabling it.

4.3 Effects of Beam Size on SEA-GUIDE

Since SEA-GUIDE uses beam search to generate the pool of candidates that it later reranks, we analyzed the effect of increasing the beam size on the SER of the final utterances. As Figure 4 shows for the ViGGO dataset, SEA-GUIDE certainly benefits from increasing the beam size from 5 to 10, but the benefit shrinks substantially (or disappears entirely, in case of T5-small) when further increased to 20. An analysis for the E2E dataset, with similar results, is presented in Appendix B.3.

[2]The 3rd component has no effect without the 2nd, so we do not consider the combination where only the 2nd is disabled.

5 Results

To maximize the performance of the models using SEA-GUIDE, the binarization thresholds (and possibly other parameters of the mention-tracking components) can be optimized for each model and dataset on the validation set. In our evaluation, however, we focused on demonstrating the effectiveness of this decoding method out of the box. That being said, even common decoding methods, such as simple beam search or nucleus sampling (Holtzman et al., 2019), usually benefit from parameter optimization (e.g., beam size, or the p-value) whenever used with a different model or dataset.

5.1 SEA-GUIDE Performance

While developing the SEA-GUIDE method we analyzed the behavior of cross-attention on both the T5-small and the BART-base model; interestingly, the decoding performs best for both with nearly the same configuration. The only difference is the 2nd component's binarization threshold (see Table 3), accounting for the fact that BART-base has 50% more attention heads than T5-small, which causes the attention weights to be more spread out.

The upper half of Table 4 compares the two models' performance with SEA-GUIDE vs. other decoding methods, as well as against three state-of-the-art baselines. As the results show, both models, when using SEA-GUIDE, significantly reduce the number of semantic errors in the generated outputs compared to using greedy search ($\approx$ 3.4 and 2.5 times in case of T5 and BART, respectively) or simple beam search ($\approx$ 1.9 times both). As expected, the slot-aligner (SA) reranking achieves even better results thanks to the handcrafted rules it relies on. In addition, the overall high automatic metric scores suggest that the fluency of utterances generated using SEA-GUIDE does not suffer.

Finally, compared to the baseline models, T5-small performs on par with the state-of-the-art DataTuner in terms of automatic metrics, yet maintains a 3.4-times lower SER. This corresponds approximately to K&M baseline's SER, whose automatic metrics, however, are significantly worse. BART-base outperforms T5-small according to most metrics, but its SER is more than double.

5.2 Cross-Model Robustness

In addition to T5-small and BART-base, we fine-tune a larger variant of each of the models, namely, T5-base and BART-large (see Appendix A.3 for model specifications), on the ViGGO dataset, and evaluate their inference performance when equipped with SEA-GUIDE. We do not perform any further tuning of the decoding parameters for these two models, only slightly lower the binarization thresholds (as we did for BART-base) to account for the models having more attention heads and layers. The thresholds we use for the 2nd and 3rd components are $\langle 0.3, 0.1 \rangle$ and $\langle 0.2, 0.05 \rangle$ for T5-base and BART-large, respectively.

The results in the lower half of Table 4 show that these two larger models, fine-tuned on ViGGO, benefit from SEA-GUIDE beyond just the effect of beam search. T5-base performs significantly better across the board than its smaller T5 variant, so there is less room for improvement to begin with. In fact, the SER using greedy search is so low (0.61%, in contrast to T5-small's 1.65%) that beam search causes it to increase. Nevertheless, SEA-GUIDE improves on both, while slightly boosting the other automatic metrics as well.

Model		BLEU	MET.	ROUGE	CIDEr	SER ↓
S2S		0.519	0.388	0.631	2.531	2.55%
DT		**0.536**	**0.394**	**0.640**	**2.700**	1.68%
K&M		0.485	0.380	0.592	2.454	***0.46%***
T5-small	GS	0.519	0.387	0.631	2.647	1.65%
	BS	0.540	0.392	0.636	2.685	0.95%
	SA	**0.541**	**0.393**	**0.637**	**2.695**	**0.24%**
	SG	**0.541**	**0.393**	**0.637**	**2.695**	0.49%
BART-base	GS	0.524	0.386	0.635	2.629	2.70%
	BS	0.544	0.393	**0.639**	2.679	2.02%
	SA	**0.547**	**0.394**	**0.639**	**2.704**	**0.39%**
	SG	0.545	0.393	**0.639**	2.698	1.07%
T5-base	GS	0.527	**0.394**	**0.639**	**2.682**	0.61%
	BS	0.534	**0.394**	0.636	2.664	0.66%
	SA	**0.536**	**0.394**	0.637	2.672	**0.19%**
	SG	**0.536**	**0.394**	0.637	2.670	0.46%
BART-large	GS	0.508	0.378	0.616	2.452	5.50%
	BS	0.535	0.391	0.628	2.612	1.78%
	SA	**0.538**	**0.394**	**0.631**	**2.659**	**0.27%**
	SG	0.533	0.391	0.627	2.613	1.41%

Table 4: Models tested on the ViGGO dataset using different decoding methods: greedy search (GS), beam search with no reranking (BS), beam search with slot-aligner reranking (SA), and SEA-GUIDE (SG). Baselines compared against are Slug2Slug (Juraska et al., 2019) (S2S), DataTuner (Harkous et al., 2020) (DT), and Kedzie and McKeown (2020) (K&M). The best results are highlighted in bold for each model. SER scores of baselines reported by the authors themselves, rather than calculated using our slot aligner, are highlighted in italics, and they do not correspond exactly to our SER results.

	Model	BLEU	MET.	ROUGE	CIDEr	SER ↓
	S2S	0.662	0.445	0.677	2.262	0.91%
	S_1^R	**0.686**	**0.453**	**0.708**	**2.370**	N/A
	K&M	0.663	**0.453**	0.693	2.308	***0.00%***
T5-small	GS	0.670	**0.454**	0.692	2.244	1.60%
	BS	0.667	0.453	**0.694**	**2.361**	2.85%
	SA	**0.675**	0.453	0.690	2.341	**0.02%**
	SG	0.675	0.453	0.690	2.340	0.04%
BART-base	GS	0.667	**0.454**	0.694	2.276	1.97%
	BS	0.670	**0.454**	**0.701**	**2.372**	3.39%
	SA	**0.680**	0.453	0.695	2.350	**0.02%**
	SG	**0.680**	0.453	0.695	2.347	0.08%
T5-base	GS	0.668	**0.459**	0.692	2.282	1.85%
	BS	0.667	0.453	**0.697**	**2.387**	3.94%
	SA	**0.682**	0.454	0.691	2.375	**0.03%**
	SG	**0.682**	0.454	0.691	2.374	0.05%

Table 5: Models tested on the E2E dataset, compared against the following baselines: Slug2Slug (Juraska et al., 2018), (S2S) S_1^R (Shen et al., 2019), and Kedzie and McKeown (2020) (K&M).

	Model	BLEU	$BLEU_R$	MET.	SER ↓	SER_E ↓
	SCG	N/A	0.308	N/A	***0.53%***	N/A
	K&R	N/A	**0.351**	N/A	N/A	1.27%
T5-small	GS	**0.367**	**0.351**	**0.325**	1.15%	1.36%
	BS	0.359	0.344	0.323	1.06%	1.19%
	SA	0.360	0.344	0.323	**0.41%**	**0.63%**
	SG	0.360	0.344	0.323	0.60%	0.85%
BART-base	GS	**0.372**	**0.356**	**0.326**	1.18%	1.17%
	BS	0.363	0.346	0.323	1.12%	1.02%
	SA	0.364	0.347	0.324	**0.40%**	**0.60%**
	SG	0.363	0.347	0.323	0.63%	0.72%

Table 6: Models tested on MultiWOZ, compared against the following baselines: SC-GPT (Peng et al., 2020) (SCG) and Kale and Rastogi (2020) (K&R).

The almost twice-as-large BART-large model performs rather poorly in our experiments, in fact, significantly underperforming its smaller variant.[3] We therefore refrain from drawing any conclusions for this model, although SEA-GUIDE offers a definite improvement in SER over simple beam search.

5.3 Domain Transferability

We achieve similar results when evaluating across domains. Table 5 shows that using SEA-GUIDE with all three models fine-tuned on E2E reduces the SER down to almost zero, with performance for the other metrics comparable to the state-of-the-art baseline.[4] In fact, SEA-GUIDE is nearly as effective at reducing errors in this dataset as the heuristic slot aligner (SA). Table 6 compares our models against two recent baselines on the MultiWOZ dataset, where the effectiveness of SEA-GUIDE on SER reduction is comparable to that on the ViGGO dataset. All in all, on both the E2E and the MultiWOZ dataset, our models equipped with SEA-GUIDE for inference perform similarly to the best baselines for both SER and the other metrics *at the same time*, whereas the baselines individually perform well according to one at the expense of the other.

5.4 Slot Error Detection Examples

Table 7 shows several utterances generated for corresponding input MRs in the video game domain, along with the errors SEA-GUIDE detected, if any. In the first example, all slots are correctly mentioned, and SEA-GUIDE agrees. This utterance was ultimately selected during reranking over the beam search's choice, "The Room is an excellent first person point-and-click puzzle game.", which has one of the genres omitted.

The second example again showcases a successful identification of all slot mentions by SEA-GUIDE, this time in an utterance where our heuristic slot aligner incorrectly found an error in the HAS_LINUX_RELEASE slot mention (presumably confused by the negation shortly before the keyword "Linux").

The third example shows that SEA-GUIDE is not confused by the difference between "excellent" (input slot value) and "amazing" (the corresponding slot realization), and it correctly identifies that the PLAYER_PERSPECTIVE slot is not mentioned.

In the fourth example, SEA-GUIDE fails to recognize the incorrect mention of the HAS_LINUX_RELEASE slot. This is the most common mistake our decoding method makes.

The final example does not contain any errors, yet the heuristic slot aligner was confused by the negation formulation (as "I wish it was") and identified the AVAILABLE_ON_STEAM slot as incorrect, while SEA-GUIDE did not. This demonstrates the robustness of SEA-GUIDE beyond exact matching. Yet despite the individual slots being mentioned semantically correctly, the generated utterance as a whole does not sound quite right. A better utterance

[3] We observed that it frequently misrepresents names, such as "Transportal Tycoon" instead of "Transport Tycoon", which we think may be the consequence of the extremely small size of the ViGGO training set relative to the model's size.

[4] We were unable to successfully train BART-large on E2E due to the memory limitations of our computational resources.

give_opinion(NAME[The Room], RATING[excellent], GENRES[indie, point-and-click, puzzle], PLAYER_PERSPECTIVE[first person])
The Room is an excellent indie point-and-click puzzle game. The first person perspective makes it easy to play. **Errors:** N/A
verify_attribute(name[Small World 2], RATING[poor], HAS_LINUX_RELEASE[yes])
You said earlier that you disliked Small World 2. Are you generally not fond of Linux games? **Errors:** N/A
request_explanation(RATING [excellent], GENRES [adventure, hack-and-slash, indie], PLAYER_PERSPECTIVE [third person])
What is it about indie hack-and-slash adventure games that you find so amazing? **Errors:** PLAYER_PERSPECTIVE
inform(NAME[RollerCoaster Tycoon], RATING[good], GENRES[simulation, strategy], PLATFORMS[Xbox, PC], AVAILABLE_ON_STEAM[no], HAS_LINUX_RELEASE[yes], HAS_MAC_RELEASE[no])
RollerCoaster Tycoon is a good simulation strategy game for Xbox and PC. It is not available on Steam, Linux, or Mac. **Errors:** N/A (incorrect)
give_opinion(NAME [Mirror's Edge Catalyst], RATING [poor], AVAILABLE_ON_STEAM [no])
Mirror's Edge Catalyst is a pretty bad game, I really wish it was available on Steam. **Errors:** N/A

Table 7: Examples of slot-mention tracking results using SEA-GUIDE. Each gray row is the input MR for the corresponding utterance in the subsequent row.

would be something along the lines of "Mirror's Edge Catalyst is a disappointment, I really wish they released it on Steam as well.".

All in all, SEA-GUIDE chooses semantically correct utterances that are fluent and adequate, except for the rare case like in the last example.

6 Discussion

In the previous section, we showed that SEA-GUIDE is highly effective at reducing semantic errors across different models and domains, and that without compromising on the generated utterances' fluency. On datasets other than E2E, it does not quite match the performance of beam search combined with our slot aligner-based reranking, but then again, the slot aligner is a hand-crafted tool with complex rules, requiring a good deal of domain knowledge, and suffering thus significantly in scalability. While these two decoding methods have a lot in common – both being based on beam search and subsequent candidate reranking – their difference lies in the identification of slot mentions; SEA-GUIDE identifies them *automatically* during the decoding, utilizing the model's cross-attention weights at each step, as opposed to relying on string-matching rules post decoding, which need to be extended for any new domains.

Despite working conveniently out of the box, SEA-GUIDE does not come with a computational overhead caveat. Performing inference on a GPU, SEA-GUIDE is a mere 11–18% slower than beam search with slot aligner-based reranking, while we observed no performance difference on a CPU (see Appendix B.4 for a detailed analysis).

6.1 Limitations of SEA-GUIDE

SEA-GUIDE's ability to recognize slot errors is limited to missing and incorrect slot mentions, which are the most common mistakes we observed models to make on the data-to-text generation task. Duplicate slot mentions are hard to identify reliably because the decoder inherently pays attention to certain input tokens at multiple non-consecutive steps (such as in the example in Figure 1b). And arbitrary hallucinations are entirely beyond the scope of this method, as there is no reason to expect cross-attention to be involved in producing input-unrelated content, at least not in a foreseeable way.

As we see in example #4 in Table 7, Boolean slots occasionally give SEA-GUIDE a hard time, as the decoder appears not to be paying a great deal of attention to Boolean slots' values throughout the entire decoding in many cases. We plan to investigate if the performance can be improved for Boolean slots, perhaps by modifying the input format or finding a more subtle slot mention pattern.

7 Conclusion

We presented a novel decoding method, SEA-GUIDE, that makes a better use of the cross-attention component of the already complex and enormous pretrained generative LMs to achieve significantly higher semantic accuracy for data-to-text NLG, while preserving the otherwise high quality of the output text. It is an automatic method, exploiting information already present in the model, but in an interpretable way. SEA-GUIDE requires no training, annotation, data augmentation, or model modifications, and can thus be effortlessly used with different models and domains.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable feedback. This research was supported by NSF AI Institute Grant No. 1559735.

References

Shubham Agarwal, Marc Dymetman, and Eric Gaussier. 2018. Char2char generation with reranking for the e2e nlg challenge. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 451–456.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *ICLR*.

Ashutosh Baheti, Alan Ritter, Jiwei Li, and William B Dolan. 2018. Generating more interesting responses in neural conversation models with distributional constraints. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3970–3980.

Andrew Chisholm, Will Radford, and Ben Hachey. 2017. Learning to generate one-sentence biographies from wikidata. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 633–642.

Kyunghyun Cho, Bart van Merriënboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*.

Ondřej Dušek and Filip Jurčíček. 2016. Sequence-to-sequence generation for spoken dialogue via deep syntax trees and strings.

Ondřej Dušek, Jekaterina Novikova, and Verena Rieser. 2018. Findings of the E2E NLG challenge. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 322–328, Tilburg University, The Netherlands. Association for Computational Linguistics.

Mihail Eric, Rahul Goel, Shachi Paul, Abhishek Sethi, Sanchit Agarwal, Shuyang Gao, Adarsh Kumar, Anuj Goyal, Peter Ku, and Dilek Hakkani-Tur. 2020. Multiwoz 2.1: A consolidated multi-domain dialogue dataset with state corrections and state tracking baselines. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 422–428.

Marjan Ghazvininejad, Xing Shi, Jay Priyadarshi, and Kevin Knight. 2017. Hafez: an interactive poetry generation system. In *Proceedings of ACL 2017, System Demonstrations*, pages 43–48.

Hamza Harkous, Isabel Groves, and Amir Saffari. 2020. Have your text and use it too! end-to-end neural data-to-text generation with semantic fidelity. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2410–2424.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, David Golub, and Yejin Choi. 2018. Learning to write with cooperative discriminators. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1638–1649.

Juraj Juraska, Kevin Bowden, and Marilyn Walker. 2019. Viggo: A video game corpus for data-to-text generation in open-domain conversation. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 164–172.

Juraj Juraska, Panagiotis Karagiannis, Kevin Bowden, and Marilyn Walker. 2018. A deep ensemble model with slot alignment for sequence-to-sequence natural language generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 152–162.

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.

Chris Kedzie and Kathleen McKeown. 2019. A good sample is hard to find: Noise injection sampling and self-training for neural language generation models. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 584–593, Tokyo, Japan. Association for Computational Linguistics.

Chris Kedzie and Kathleen McKeown. 2020. Controllable meaning representation to text generation: Linearization and data augmentation strategies. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5160–5185.

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. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of*

the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1412–1421.

Haitao Mi, Baskaran Sankaran, Zhiguo Wang, and Abe Ittycheriah. 2016. Coverage embedding models for neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 955–960.

Feng Nie, Jin-Ge Yao, Jinpeng Wang, Rong Pan, and Chin-Yew Lin. 2019. A simple recipe towards reducing hallucination in neural surface realisation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2673–2679.

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2017. The e2e dataset: New challenges for end-to-end generation. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 201–206.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *ACL*.

Baolin Peng, Chenguang Zhu, Chunyuan Li, Xiujun Li, Jinchao Li, Michael Zeng, and Jianfeng Gao. 2020. Few-shot natural language generation for task-oriented dialog. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 172–182.

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*, 21:1–67.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Sheng Shen, Daniel Fried, Jacob Andreas, and Dan Klein. 2019. Pragmatically informative text generation. 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 4060–4067.

Xiaoyu Shen, Ernie Chang, Hui Su, Cheng Niu, and Dietrich Klakow. 2020. Neural data-to-text generation via jointly learning the segmentation and correspondence. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7155–7165.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 76–85.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6000–6010. Curran Associates Inc.

Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. 2015. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4566–4575.

Tsung-Hsien Wen, Milica Gašić, Dongho Kim, Nikola Mrkšić, Pei hao Su, David Vandyke, and Steve Young. 2015a. Stochastic language generation in dialogue using recurrent neural networks with convolutional sentence reranking. In *SIGDIAL Conference*.

Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015b. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *EMNLP*.

	Training	Validation	Test
ViGGO	5,103	714	1,083
E2E	42,063	4,672	4,693
MultiWOZ	55,951	7,286	7,293

Table 8: Overview of the dataset partitions.

A Appendix

A.1 Additional Dataset Details

Table 8 shows the number of examples in the training, validation and test partitions of all the datasets used in the evaluation of the SEA-GUIDE method.

A.2 Data Preprocessing

When preprocessing input meaning representations (MRs) before training a model or running inference, we first parse the dialogue act (DA) types, if present, and all slots and their values from the dataset-specific format into an intermediate list of slot-and-value pairs, keeping the original order. Although typically indicated in the MR differently from slots, we treat the DA type as any other slot (with the value being the DA type itself, and assigning it the name "intent").

Next, we rename any slots that do not have a natural-language name (e.g., "priceRange" to "price range", or "has_mac_release" to "has Mac release"). Slot values are left untouched. We do this to take advantage of pretrained language models' ability to model the context when the input contains familiar words, as opposed to feeding it code names with underscores and no spaces.

Finally, we convert the updated intermediate list of slots and their values to a string. The '|' symbol is used for separating slot-and-value pairs from each other, while the '=' is used within each pair to separate the value from the slot name. The result for an MR from ViGGO can look as follows:

```
intent = request explanation
| rating = poor | genres =
vehicular combat | player
perspective = third person
```

A.3 Model and Training Parameters

The pretrained models that we fine-tuned for our experiments are the PyTorch implementations in the Hugging Face's Transformers[5] package. The models' sizes are indicated in Table 9.

We trained all models using a single Nvidia RTX 2070 GPU with 8 GB of memory and CUDA version 10.2. The training parameters too are summarized in Table 9. For all models, we used the AdamW optimizer with a linear decay after 100 warm-up steps. The maximum sequence length for both training and inference was set to 128 for ViGGO and E2E, and 160 for MultiWOZ.

	Layers	Heads	Hidden state size	Total parameters
T5-small	6+6	8	512	$\approx$ 60M
BART-base	6+6	12	768	$\approx$ 139M
T5-base	12+12	12	768	$\approx$ 220M
BART-large	12+12	16	1024	$\approx$ 406M

Table 9: Overview of the model specifications.

	Batch size	Learning rate	Epochs
T5-small	32/64/64	2×10^{-4}	20/20/30
BART-base	32/32/32	1×10^{-5}	20/20/25
T5-base	16/16/ –	3×10^{-5}	20/20/ –
BART-large	16/ – / –	4×10^{-6}	20/ – / –

Table 10: Overview of the training parameters used in our experiments. Batch size and the number of epochs are indicated per dataset (ViGGO/E2E/MultiWOZ).

A.4 Evaluation Metric Calculation

The four non-SER automatic metrics that we report in our results (i.e., BLEU, METEOR, ROUGE-L, and CIDEr) are calculated using the E2E evaluation script[6] developed for the E2E NLG Challenge (Dušek et al., 2018). We also verified that the single-reference BLEU score calculation in the E2E script corresponds to that in the SacreBLEU[7] Python package. As a result, BLEU scores calculated either way are directly comparable.

To ensure a fair comparison with the MultiWOZ baselines (Peng et al., 2020; Kale and Rastogi, 2020), we additionally report BLEU scores calculated using the RNNLG evaluation script[8], which their respective authors used in their own evaluation. We denote it BLEU_R in our result tables. Moreover, Kale and Rastogi (2020) calculated SER on utterance level, rather than slot level, and that using *exact* slot value matching in the utterance. We thus wrote a script to also perform this type of naive SER evaluation, in addition to our slot aligner-based SER evaluation. We report its results as SER_E.

[5] https://huggingface.co/transformers/

[6] https://github.com/tuetschek/e2e-metrics

[7] https://pypi.org/project/sacrebleu/

[8] https://github.com/shawnwun/RNNLG/

B Additional SEA-GUIDE Evaluation

B.1 Slot Mention Tracking Details

In order to be able to take advantage of the attention weight distribution patterns, the decoder needs to be aware of which input token span corresponds to which slot. To this end, we parse the input MRs on-the-fly – which is trivial given the structured nature of MRs – as each batch is being prepared for inference, and create a list of slot spans for each MR in the batch.[9] In fact, we indicate the spans for slot names and slot values separately, and for list-values down to individual list elements, for a higher specificity. Since Boolean slot mentions are tracked by their name rather than value, we also indicate for each slot whether it is Boolean or not. This information can be provided explicitly to the data loader, otherwise it is automatically inferred from the dataset's ontology based on all the possible values for each slot.

Note that, although our data preprocessing converts DA type indications in the MRs to the same format as slots (see any of the left columns in Figure 1), we exclude them from the slot-span lists, as they are not actual content slots to be tracked. Separator tokens (such as '|' or '=') present in the preprocessed MR are not included in the spans, and are, as a result, ignored during the slot mention tracking.

B.2 Parameter Tuning for T5

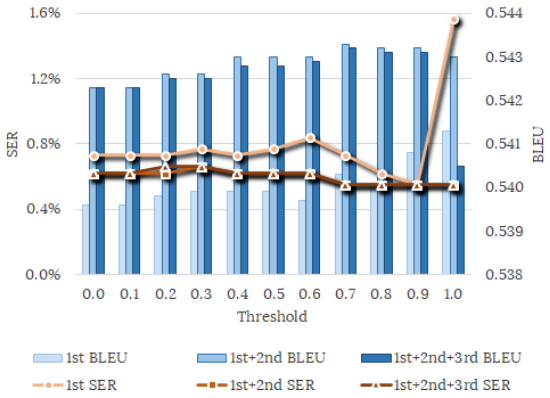
When optimizing the mention-tracking components' parameters for T5-small, we observe similar trends as with BART-base (see Figure 5). One difference is that enabling the 2nd component not only significantly increases the BLEU score, but also lowers the SER, while the 3rd component appears to only have a negligible effect (see Figure 5a).

B.3 Effects of Beam Size on E2E

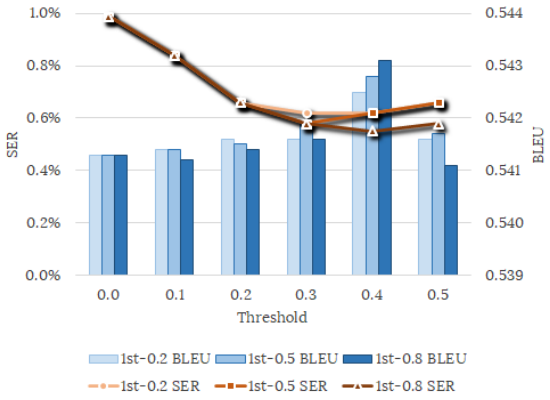
On the E2E dataset, decoding using SEA-GUIDE is even more effective in reducing SER than on ViGGO. Across all beam sizes, its performance is comparable to beam search with slot aligner reranking, and there is also only a limited gain from increasing the beam size to 20 (see Figure 6).

It is worth noting that, using beam search with no reranking, the SER dramatically increases with the increasing beam size. This is likely caused by the relatively heavy semantic noise in the E2E

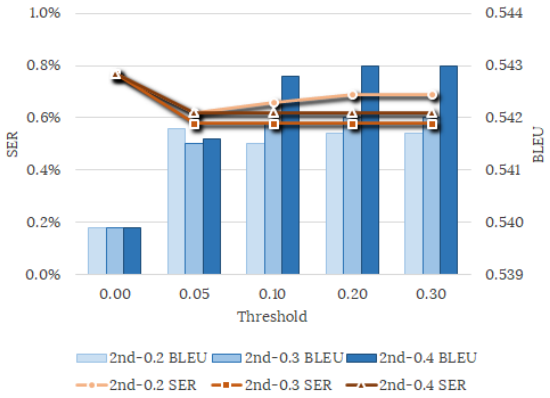
[9] This is done on token level, and the result varies thus from model to model depending on its tokenizer.



(a) Threshold optimization for the 1st component (verbatim mentions), with the other components enabled or disabled. When enabled, the 2nd component's threshold was fixed at 0.3, and that of the 3rd at 0.1. Note that the threshold of 1.0 is equivalent to the 1st component being disabled, as attention weights are in the [0.0, 1.0] range.



(b) Threshold optimization for the 2nd component (paraphrased mentions), with the 1st component's threshold of 0.2, 0.5 and 0.8, and that of the 3rd component fixed at 0.1.



(c) Threshold optimization for the 3rd component (unrealized mentions), with the 2nd component's threshold of 0.2, 0.3 and 0.4, and that of the 1st component fixed at 0.5.

Figure 5: Effects of different parameter configurations of the 3 mention-tracking components on SER and BLEU of utterances generated by T5-small fine-tuned on ViGGO.

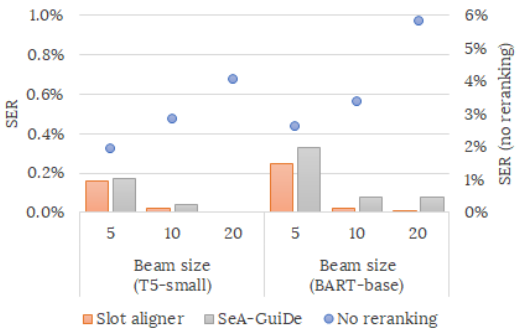


Figure 6: The effect of different beam size on the SER using different reranking methods on the E2E dataset. With greedy search decoding, the SER is 1.60% and 1.97% for T5 and BART, respectively.

training set, resulting in more slot errors in the generated utterances the less greedy the decoding is. Some form of semantic guidance is thus all the more important for the model in this scenario.

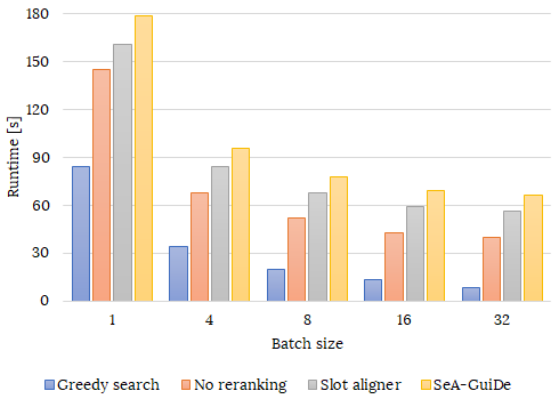
B.4 Inference Performance

In order to assess the computational overhead the SEA-GUIDE method introduces during inference, we measure the inference runtime of the T5-small model fine-tuned on ViGGO. For all beam search-based methods (including SEA-GUIDE), the beam size was set to 10, and early stopping was enabled.

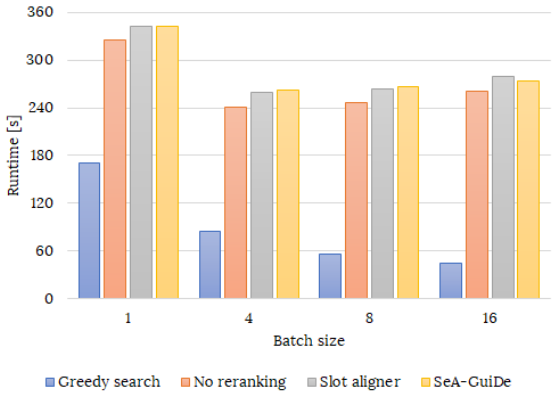
The results in Figure 7a show a distinct but expected overhead across all batch sizes when running inference on a GPU. The overall increase in runtime is 11–18% over beam search with slot aligner-based reranking, which is the method computationally most similar to SEA-GUIDE, as it too involves reranking on top of beam search. The slot aligner-based reranking itself adds a constant amount of 16 seconds on top of simple beam search, which corresponds to an 11-40% increase for the range of batch sizes in the plot.

When performing the same inference on a CPU, on the other hand, the overhead SEA-GUIDE introduces to beam search is no greater than that of the slot aligner-based reranking (see Figure 7b). This suggests that further optimization of SEA-GUIDE for GPU, especially by minimizing the communication between the GPU and the CPU during the decoding, could bring the overhead of SEA-GUIDE inference on a GPU down to the same level as that of the slot aligner-based reranking.

Considering the large improvement in semantic accuracy the SEA-GUIDE method delivers in the tested models, we deem the observed computational overhead reasonable and acceptable.



(a) Inference using a GPU (RTX 2070 with 8 GB of memory).



(b) Inference using a CPU (8-core Ryzen 7 2700X with 32 GB of RAM).

Figure 7: Runtime of T5-small performing inference on the ViGGO test set using different decoding methods and batch sizes. "No reranking" stands for simple beam search, while "Slot aligner" denotes beam search with slot aligner-based reranking. Model and data loading is excluded from the runtimes.

C Slot Aligner Details

For the purposes of the slot aligner, we classified slots into five general categories (*Boolean*, *numeric*, *scalar*, *categorical*, and *list*), covering the most common types of information MRs typically convey in data-to-text NLG. Each of these categories has its own method for extracting a slot mention from an utterance, generalized enough to be applicable across all slots in the category. This design allows for a straightforward extension of the slot aligner to a new domain, as it merely needs to be indicated which of the five categories each of the slots in the new domain belongs to. Optionally, it can be provided a simple dictionary of common alternatives for specific slot values, which tends to increase the slot aligner's performance.

Although a decreased matching accuracy – es-

pecially for rare slot realizations – is a trade-off for the scalable design, the slot aligner's typical application are generated model outputs, which get evaluated for semantic errors. There the slot aligner is not likely to encounter rare slot realizations frequently, if at all, due to the generalizing properties of neural NLG models. The rapid adaptability of the slot aligner to a new domain, on the other hand, is a very valuable feature.

C.1 Boolean Slots

Boolean slots take on binary values, such as "yes"/"no" or "true"/"false". Their realization in an utterance thus typically does not contain the actual value of the slot, but instead a mention of the slot name (e.g., "is a family-friendly restaurant" for FAMILYFRIENDLY[yes], or "not supported on Mac" for HAS_MAC_RELEASE[no]). Therefore, extracting a Boolean slot mention boils down to the following two steps: (1) finding a word or a phrase representing the slot, and (2) verifying whether the representation is associated with a negation or not.

The first step is straightforward, and only requires a list of possible realizations for each Boolean slot. This list rarely contains more than one element, which is the "stem" of the slot's name (e.g., "linux" for "HAS_LINUX_RELEASE"). It can thus be populated trivially for most of the new Boolean slots. And if a Boolean slot can have multiple equivalent realizations (such as "child friendly" or "where kids are welcome" for the slot FAMILYFRIENDLY), they are typically not numerous and can be listed manually. Having a list of stems (we refer to all the equivalent realizations of a slot collectively as "slot stems"), the utterance is scanned for the presence of each of them in it. If one is found, we go to the second step. A slot mention is decided to be negative if a negation cue is found to be modifying the slot stem in the utterance, and without a contrastive cue in between. It is decided to be positive if no negation cue is present within a certain distance of the stem, or there is a contrastive cue in between (see examples in Table 11).

C.2 Numeric Slots

Slots whose value is just a number (such as RELEASE_YEAR in ViGGO, or CHOICE in MultiWOZ) are in general not handled in any special way, and the value is simply matched directly in the utterance. However, there are certain numeric slot types that benefit from additional preprocessing: (1) those with a unit, and (2) years. When a numeric slot represents a year, the slot aligner generates the common abbreviated alternatives for the year (e.g., "'97" for the value "1997") that it tries to match in case the original value is not found in the utterance.

#1	There's *no* Linux release or multiplayer, **but** there is Mac support.
#2	**Though** it's *not available* on Linux, it does have a Mac release as well.
#3	It is available on PC and Mac **but** *not* Linux, and it can be found on Steam.

Table 11: Examples of contrastive phrases involving Boolean slots. Underlined are the stems of the Boolean slots for which the polarity is questioned. Note that in all 3 examples the mention is positive, despite the presence of contrast and negation distractors.

CUSTOMER RATING (E2E)	RATING (ViGGO)	Alternative expressions
low	poor	*bad, lacking, negative,...*
average	average	*decent, mediocre, okay,...*
-	good	*fun, positive, solid,...*
high	excellent	*amazing, fantastic, great,...*

Table 12: An example of value mapping between two similar scalar slots in the restaurant and video game domains.

C.3 Scalar Slots

Similarly to Boolean slot aligning, scalar slot aligning consist of two steps. The first one is the same, i.e., finding a word or a phrase representing the slot (which we refer to as "stem" in this case too, in order to maintain consistency). In the second step, however, the slot aligner looks for the slot's value, or its equivalent, occurring within a reasonable distance from the slot stem. The optional soft alignment mode skips the second step as long as a slot stem is matched in the first step.

We assume that scalar slots, even across different domains, will often have values that can be mapped to each other, as long as they are on the same or a similar scale (see Table 12). For each scalar slot, the slot aligner refers to a corresponding dictionary for possible alternative expressions of its value. With the above assumption, it is sufficient to have one dictionary per scale, or type of scale, which can be reused for similar scalar slots in different domains. The dictionaries can be quickly populated with synonyms of the values of a given scale (see the last column in the table), and thus

do not necessarily require manual additions every time the system is used with a new domain. Some alternative expressions might be suitable for scalar slots in some domains better than others, but that will not be an issue in most cases, since, being synonymous, they are not likely to cause conflicts, and the slot aligner will simply not encounter certain alternative expressions in certain domains.

C.4 Categorical Slots

Categorical slots can take on virtually any value. Nevertheless, for each such slot the values typically come from a limited, although possibly large, set of values. For instance, in the E2E dataset, the FOOD slot has 7 possible values, such as "Italian" and "Fast food", but technically it could take on hundreds of different values representing all of the cuisines of the world. Some values can be single-word, while others can have multiple words (e.g., "restaurant" and "coffee shop" as possible values for the EATTYPE slot). Due to this huge variety in possible values of categorical slots, the aligning methods need to remain very general.

Besides exact matching of the value in the utterance, the slot aligner can be instructed to perform the matching in three additional modes, besides exact, increasing its robustness while maintaining scalability. The four modes of aligning the slot with its mention work as follows:

- **Exact** - slot mention is identified only if it matches (case-insensitive) the slot value verbatim;
- **All words** - slot mention is identified if each of the value's tokens is found in the utterance, though they can be in an arbitrary order and they can be separated by other words;
- **Any word** - slot mention is identified by matching any of the value's tokens in the utterance;
- **First word** - slot mention is identified by matching just the value's first token in the utterance.

Note that for single-word values all four modes give the same result. The three non-exact modes offer different approaches to soft alignment for categorical slots. The choice may depend on the particular slot, and the mode can thus be specified for each slot separately, while by default the slot aligner operates in the exact-matching mode.

MR
inform(NAME [**BioShock**], DEVELOPER [**2K Boston**], GENRES [**action-adventure, role-playing, shooter**], HAS_MULTIPLAYER [**no**], PLATFORMS [**PlayStation, Xbox, PC**], HAS_LINUX_RELEASE [**no**], HAS_MAC_RELEASE [**yes**])
Reference utterance
Developed by **2K Boston**, **BioShock** is a **single-player shooter** game that will have you **role-playing** through a well constructed **action-adventure** narrative. It is **available for PlayStation**, **Xbox**, **Mac** and **PC**, but is **not available for Linux**.
Slot alignment
(**13:** DEVELOPER) (**25:** NAME) (**39:** HAS_MULTIPLAYER) (**53:** GENRES) (**174:** PLATFORMS) (**191:** HAS_MAC_RELEASE) (**228:** HAS_LINUX_RELEASE)

Table 13: An example from ViGGO that involves list slots. Notice how the individual value item mentions can be scattered across an entire sentence in a natural way. The bottom section indicates the slot mention positions determined by the slot aligner, given as the number of characters from the beginning of the utterance.

Similarly to Boolean and scalar slots, the slot aligner can search for alternative expressions of a value, if provided in the corresponding dictionary. The alternative matching is, however, more flexible here, as the alternatives in the dictionary can be multi-part, in which case the slot aligner tries to match all the parts (words/tokens/phrases) provided in the form of a list.

C.5 List Slots

A list slot is similar to a categorical slot, the only difference being that it can have multiple individual items in its value. Two instances of a list slot, namely GENRES and PLATFORMS, can be seen in the example from the ViGGO dataset in Table 13.

The aligning procedure for list slots thus heavily relies on that of categorical slots. In order to align a list slot with the corresponding utterance, the slot aligner first parses the individual items in the slot's value. It then iterates over all of them and performs the categorical slot alignment, as described in the previous section, with each individual item. Considering the items can be scattered over multiple sentences, the slot aligner considers the position of the leftmost mention of an item as the position of the corresponding list slot.