Data-to-text Generation with Verification and Correction Prompting

Anonymous ACL submission

Abstract

Small language models like T5 excel in generating high-quality text for data-to-text tasks, offering adaptability and cost-efficiency compared to Large Language Models (LLMs). However, they frequently miss keywords, which is considered one of the most severe and common errors in this task.

In this work, we explore the potential of using feedback systems to enhance semantic fidelity in smaller language models for data-to-text generation tasks, through our Verification and Correction Prompting (VCP) approach.

In the inference stage, our approach involves a multi-step process, including generation, verification, and regeneration stages. During the verification stage, we implement a simple rule to check for the presence of every keyword in the prediction. Recognizing that this rule can be inaccurate, we have developed a carefully designed training procedure, which enabling the model to incorporate feedback from the error-indication prompt effectively, despite its potential inaccuracies.

The VCP approach effectively reduces the Semantic Error Rate (SER) while maintaining the text's quality.

1 Introduction

give_opinion(NAME [**SpellForce 3**], RATING [**poor**], GENRES [**real-time strategy, role-playing**], PLAYER_PERSPECTIVE [**bird view**])
I think that **SpellForce 3** is **one of the worst games** I've ever played. Trying to combine the **real-time strategy** and **role-playing** genres just doesn't work, and the **bird's eye view** makes it near impossible to play.

Figure 1: The ViGGO dataset (Juraska et al., 2019).

Data-to-text generation aims to convert structured data into coherent, human-readable text, as shown in Figure 1. It has a broad range of practical applications, including report generation, automated journalism, data visualization, and dialogue systems, or it can be used as intermediate steps in large projects. In these applications, the input can consist of various types of data, such as tables, graphs, or raw data. It is worth noting that data-to-text generation is a controlled form of text generation, where the output must be coherent with the input and maintain semantic accuracy.

Fine-tuning pre-trained small models, such as T5 (Raffel et al., 2020), which are more efficient compared to LLMs, is often sufficient for many data-to-text generation tasks, as these tasks do not require strong reasoning skills. The main challenge lies in ensuring adherence to instructions and accurately replicating specific text styles. One of the most severe and frequent problems is omissions, the absence of crucial keywords (Yin and Wan, 2022). For instance, in the first example from Figure 1, if the prediction omits the name 'SpellForce 3', then the prediction has one slot error. In this paper, we introduce the 'slot error rate (SER),' which quantifies the rate of missing keywords.

Several research works aim to reduce the SER, including the copy mechanism (Rebuffel et al., 2019; Puduppully et al., 2019), template-based generation (Kale and Rastogi, 2020; Mehta et al., 2022), planning-then-generate (Xu et al., 2021; Su et al., 2021; Kasner and Dusek, 2022), and post-editing (Jolly et al., 2022; Balachandran et al., 2022). These methods often rely on strict rules and can effectively reduce SER but may sacrifice text fluency. Techniques such as those by (Juraska and Walker, 2021; Seifossadat and Sameti, 2023) guide attention behavior, leading the model to make more accurate generations. Such methods are flexible, thus reducing SER while maintaining text fluency.

Regeneration according to feedback (Madaan et al., 2023; Xue et al., 2023) has recently gained popularity, predominantly in LLMs. This approach requires an accurate feedback system to generate natural language feedback prompts. However, smaller language models, focused on efficiency, may not interpret these feedback prompts accurately and often lack a precise verifier. Employing an accurate verifier, such as an LLM or meticulously handcrafted rules, would contradict the original goal of prioritizing efficiency. We aim to explore the feasibility of using a feedback system with a trivial verifier to enhance the semantic accuracy of a smaller model in data-to-text generation.

Inspired by previous works, we propose a prompt-based, test-time correction pipeline, VCP, designed to encourage the model to include missed slots identified by a slot error checker while maintaining text quality. The overall inference process is illustrated in Figure 2. Initially, the fine-tuned T5 model receives the input slots and generates initial predictions. The slot error checker then verifies whether any slots are missing from the output. If a slot is missing, we label the corresponding error-correcting prompts along with the missed input value. During the regeneration process, these error-correcting prompts guide the fine-tuned T5 model to include the omitted slot value in its subsequent prediction. An example of this is provided in Table 1.

To enable the aforementioned error-correcting regeneration process, it is necessary to train error-correcting prompts that encourage the model to include slots it previously missed. Since the slot error checker relies on trivial rules, the trained prompts must guide the T5 model in such a way that it does not alter its prediction when the prompt is mislabeled. The training process for these prompts is outlined in the training section of Figure 2. Specifically, in the Data Generation process, a data generator is used to create unseen prompted inputs along with their corresponding ground truths. This serves to construct both the prompt initialization training dataset and the prompt training dataset. During training, we first train the prompt initialization and then fine-tune the prompt embedding based on the initialized prompt. These error-correcting prompts are designed to direct the fine-tuned T5 model to include the labeled slot values it previously missed in its predictions. During training, the model and the prompts are exposed to scenarios where slots are mislabeled by error-correcting prompts, teaching them to disregard inaccurate labels.

Our method achieves a lower SER while maintaining competitive text fluency compared to other methods.

2 Related Works and Comparative Analysis

In data-to-text generation tasks, the omission of keywords (slot error) is identified as the most severe and frequent error (Yin and Wan, 2022).

Several strategies have been developed to address this issue, each falling into distinct categories.

The first strategy involves using a strict generation process to ensure the inclusion of keywords. This includes methods such as the copy mechanism (Rebuffel et al., 2019; Puduppully et al., 2019), template-based generation methods (Kale and Rastogi, 2020; Mehta et al., 2022), and plan-then-generate approaches (Xu et al., 2021; Su et al., 2021; Kasner and Dusek, 2022). These methods enforce a model's generation to strictly adhere to the input structure. While they effectively minimize slot errors, they suffer from reduced text fluency due to their inherent inflexibility.

The second strategy involves using a post-editing approach. (Jolly et al., 2022) search for missing keywords and find the best position to insert the phrase containing these keywords. This approach is not directly comparable to our method because they only conducted experiments in a few-shot setting. (Balachandran et al., 2022) adversarially train an error correction network to correct factual errors in summarizing tasks. The error correction training dataset is constructed by replacing correct factual words with incorrect ones. In data-to-text generation, missing even one keyword can disrupt the entire sentence, thus this method cannot be directly applied to data-to-text generation.

The third strategy involves guiding the attention behavior. (Juraska and Walker, 2021) manually identified three attention patterns associated with semantic errors. They created a script to automatically adjust the beam search scores according to these three attention patterns during inference. By adding a dynamic memory module to the attention-based network, DM-NLG (Seifossadat and Sameti, 2023) can store previously generated words, thus better guiding the generation process to include key information. These two approaches reduce SER while maintaining the quality of text generation.

There have been increasing works utilizing feedback systems for generating better predictions (Madaan et al., 2023; Xue et al., 2023; Peng et al., 2023; Shridhar et al., 2023b,a). They are mostly used for reasoning tasks and have an inference process similar to our work. These feedback systems

Step1: Initial Prediction	
Input	recommand(name(Tom Clancy], release_year[1999], has_linux_release[yes])
T5 predictions	Since you're into Linux games, you heard of Tom Clancy?
Step2: Verification	
Find slot errors	1999 ×, Tom Clancy, Linux
Label missing slots	recommand(release_year[<token1><token2><token3>1999], name(Tom Clancy], has_linux_release[no])
Step3: Regeneration	
Prompted Input	recommand(release_year[<token1><token2><token3>1999], name(Tom Clancy], has_linux_release[no])
Send to T5 to generate prediction	Since you're into Linux games, have you heard of Tom Clancy which is released in 1999?

Table 1: The inference process comprises three steps: 1. We utilize the fine-tuned T5 model to generate an initial prediction. 2. The slot error checker is then deployed to ascertain the presence of any slot errors. If such errors are detected, we label the error-correcting prompts to highlight the location of potential slot errors. 3. Lastly, we reintroduce the prompted input to the fine-tuned T5 model for regeneration. The tokens(error-correcting prompts) have the capacity to alter the regenerated outputs, ensuring the inclusion of previously missed slots.

often use a Large Language Model (LLM), a prior knowledge base, or some rules to verify if there are mistakes in the initial generation. An LLM will then read the instructions returned by the verifier and regenerate the output accordingly.

Our approach is distinct from previous feedback systems in that: 1. Our verification process is efficient and trivial. It does not require prior knowledge or a LLM. 2. Our prompt feedback can guide smaller models that are not capable of responding to natural language feedback. 3. The prompt is capable of handling inaccurate feedback. 4. Our VCP can maintain text generation styles. Our work differs from previous post-editing models in that instead of applying a post-editing model to modify the details of the initial output, we use error-indicating prompts to guide the model in regenerating the output. Regenerating the entire output allows the model to reorder the sentence structure or infill the missing prompts in a flexible way according to the previously missed slot, resulting in more fluent and consistent text output.

3 Methodology

Figure 2 provides a comprehensive illustration of our method, detailing both the inference and training stages involved in the process.

3.1 Inference

An inference example is presented in Table 1, encompassing three steps: initial generation, verification, and regeneration. Initially, we input the testing samples into the fine-tuned T5 model, which was trained with the original training dataset, to generate the initial prediction. During the verification step, the Slot Error Checker, employing simple rules, examines whether any slots from the input are missing in the output, thereby identifying slot errors. If any potential errors are detected, we introduce error-correcting prompts adjacent to the positions of the unmentioned slots in the input. Lastly, in the regeneration step, the prompted inputs are fed back into the fine-tuned T5 model. These error-correcting prompts guide the T5 model to incorporate the previously omitted slots during the regeneration process.

3.2 Slot Error Checking

The Slot Error Checker verifies the presence of slot errors. For non-boolean slot value, we simply verify that all slot values are included in the prediction. For boolean slot value pairs, we do not examine whether the actual slot value, that is, *yes* or *no*, is present in the prediction. Instead, we focus on whether the noun part of the slot names, identified by part-of-speech (POS) tagging—such as *linux* from *has_linux_released*, *mac* from *has_mac_released*, or *steam* from *available_on_steam*, is mentioned in the predictions. To determine if the slot-value pairs are boolean, we check whether the slot value is *yes* or *no*. The checking process is straightforward, and we have not implemented any domain-specific knowledge in the slot error checkers.

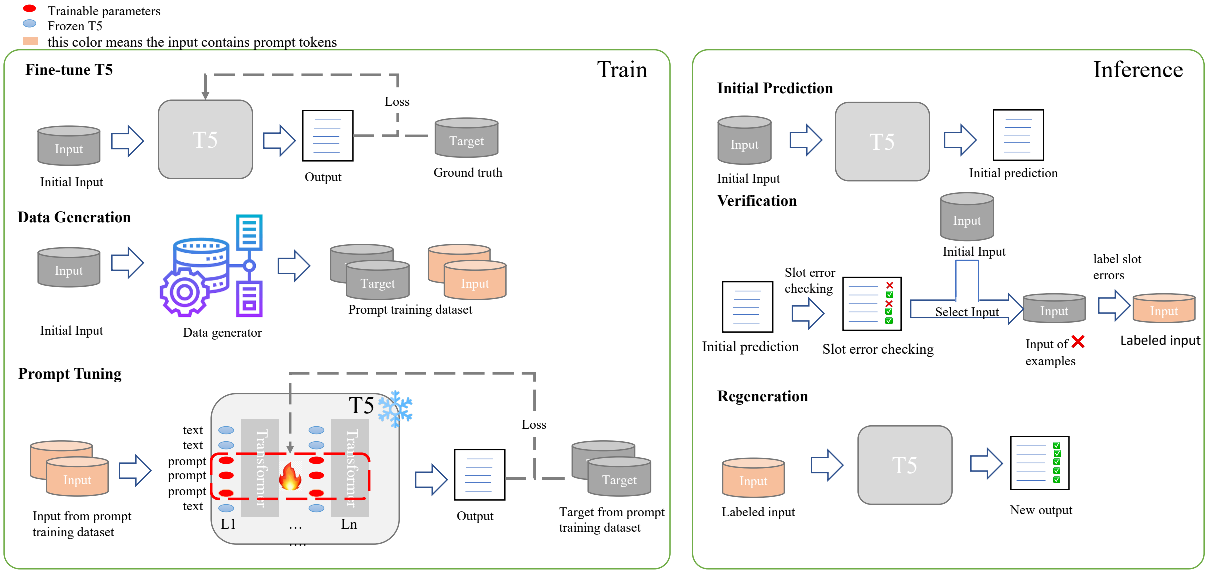

Figure 2: The workflow is comprised of train section and inference section. When training, we first fine tune a T5 model, then use data generator to generate prompt training dataset. Lastly use prompt tuning to teach error-correcting prompts how to improve the semantic coverage in T5's prediction. The inference section illustrates the overview of the initial prediction, verification, and regeneration process. Please refer to Table 1 for more detailed insights.

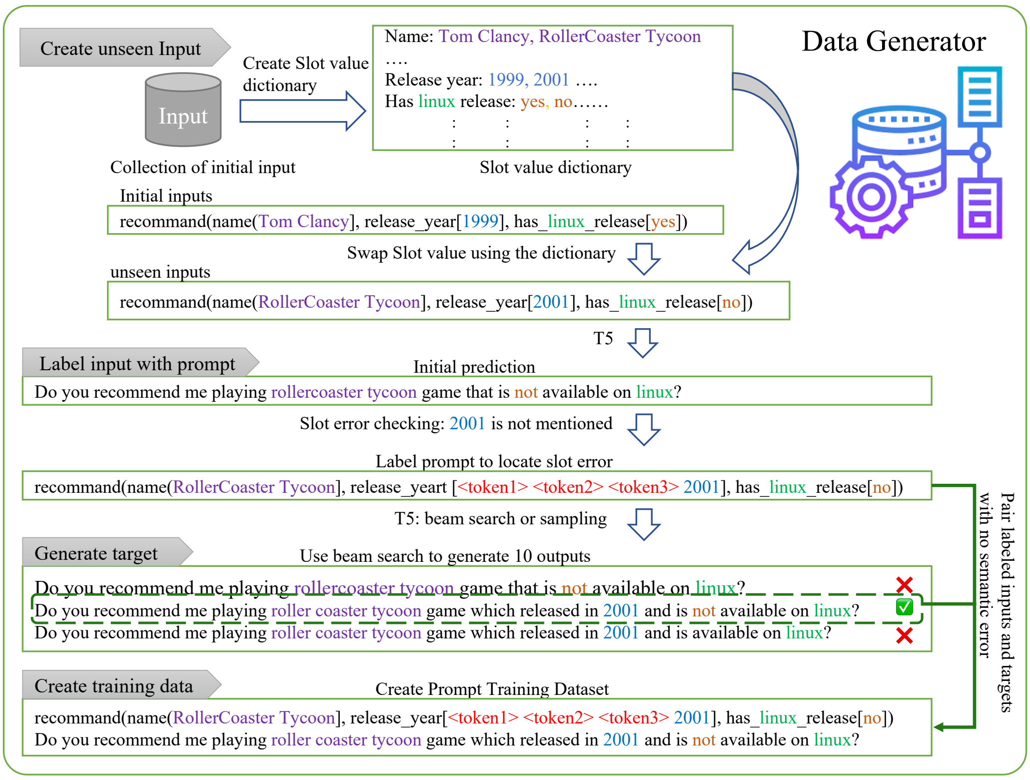

Figure 3: The workflow for generating datasets for training error-correcting prompts.

3.3 Training

The training procedure, as illustrated in Figure 2, begins by fine-tuning a T5 model using the original training dataset. This step establishes a well-initialized base model. Subsequently, a prompt training dataset is created for learning error-correcting prompts. Finally, while keeping the fine-tuned T5 model frozen, the error-correcting prompts are trained on the prompt training dataset to enhance their ability to guide the language model in integrating slot values that were previously missed.

Data Generation: Since the training data has already been exposed to the T5 model during the initial fine-tuning, we cannot use the same dataset to learn the error-correction prompts. Therefore, a new training set is generated for prompt learning, as depicted in Figure 3. Specifically, the data generation process (data generator) creates input-output pairs. The candidate input is generated by replacing the slot values in an input from the original training set with other values randomly sampled from the possible values for each slot. For example, "*recommend(name[Tom Clancy], release_year[1999], has_linux_release[yes])*" comprises the intention "*recommend*", slot names "*release_year*, *name*, and *has_linux_release*", and slot values "*1999*, *Tom Clancy*, and *yes*." We use a slot value dictionary, created from grouping all unique values corresponding to the same slot name in the training set, for this replacement. After slot value replacement, we could generate an unseen input like "*recommend(name[RollerCoaster Tycoon], release_year[2001], has_linux_release[no])*." These candidate inputs are then fed into the fine-tuned T5 model to generate initial predictions. The Slot Error Checker is applied to identify the parts of the inputs with slot errors, which are subsequently marked with an error-correcting prompt.

Then, ground-truth output for these prompted unseen inputs needs to be generated. We pass the collected prompted unseen inputs into the fine-tuned T5 model to produce 10 predictions using beam search. These predictions are then subjected to Slot error checking. The prediction that is free of slot errors is selected as the ground truth. In scenarios where multiple outputs from beam search are free of slot errors, the output with the highest probability, as determined by the beam search, is chosen as the ground truth.

Prompt Tuning. Once sufficient input-output pairs are generated, we fine-tune the error-correcting prompts while keeping the T5 model fixed. The training process is designed to learn error-correcting prompts that guide the T5 model to produce outputs without missing slot values. As the prompt training dataset contains examples that are correctly or wrongly labeled, the prompts learn how to handle these situations during training. For instance, the slot 'RATING [poor]' is tagged with an error indication prompt because it does not align with the reference 'one of the worst games.' However, these error indication prompts do not affect the prediction during regeneration. This characteristic makes our method more robust compared to previous post-editing methods that rely on strict rules.

In our design, we train deep prompts (P-Tuning v2 (Liu et al., 2022)). Specifically, we use 3 error-correcting prompts for T5-base and 6 error-correcting prompts for T5-small. The trainable prompts are added to each layer in T5, encompassing the word embeddings of the error-correcting prompts and the key-value embeddings in every layer.

Prompt Initialization In addition to the workflow shown in Figure 2, our ablation study finds the advantages of introducing a prompt initialization phase. This phase trains a robust initial embedding, ensuring that text generation quality is unaffected by prompt insertion. More details are described in the appendix.

4 Experiment

We compare our VCP method with various other approaches on the E2E (Novikova et al., 2017) and ViGGO (Juraska et al., 2019) datasets. Our primary comparison is with the two methods that guide attention behavior, SEA-GUIDE (Juraska and Walker, 2021) and DM-NLG (Seifossadat and Sameti, 2023), because they perform the best. The other methods included in our comparison are K&M (Kedzie and McKeown, 2020), which utilizes data augmentation, and DT (Harkous et al., 2020), which employs a generation-reranking approach. S2S (Juraska et al., 2019) is the baseline mentioned in the original ViGGO dataset paper.

4.1 Dataset and Evaluation Metrics

The experiments are conducted on the E2E and ViGGO datasets. The E2E dataset (Novikova et al., 2017), specifically designed for the restaurant domain, offers a data-driven approach for end-to-end natural language generation system training. The ViGGO dataset (Juraska et al., 2019) targets open-domain dialogue systems in video game topics, covering 9 generalizable and conversational dialogue act types.

Our system's performance is assessed using a comprehensive set of metrics. For non-semantic error evaluation, we use BLEU (Papineni et al., 2002), METEOR (Lavie and Agarwal, 2007), ROUGE (Lin, 2004), CIDEr (Vedantam et al., 2015), which are assessed using the E2E evaluation

Model	BLEU	MET.	ROUGE	CIDEr	SER ↓	SLSER
T5-small$_{beamsearch}$ baseline	53.2 ± 0.54	0.392	0.637	2.652	$0.89 \pm 0.096\%$	3%
T5-base$_{beamsearch}$ baseline	53.1 ± 0.28	0.393	0.635	2.655	$0.60 \pm 0.13\%$	2%
S2S(Juraska et al., 2019)	51.9	0.388	0.631	2.531	2.55%	-
DT(Harkous et al., 2020)	53.6	0.394	0.640	2.700	1.68%	-
K&M(Kedzie and McKeown, 2020)	48.5	0.380	0.592	2.454	*0.46%*	-
SEA-GUIDE$_{T5-small}$(Juraska and Walker, 2021)	53.2 ± 0.53	0.392	0.637	2.693	$0.7 \pm 0.097\%$	2.0%
SEA-GUIDE$_{T5-base}$(Juraska and Walker, 2021)	53.2 ± 0.30	0.393	0.635	2.658	$0.51 \pm 0.10\%$	1.7%
VCP$_{T5-small}$	52.6 ± 0.51	0.392	0.632	2.628	$\mathbf{0.41} \pm 0.085\%$	1.4%
VCP$_{T5-base}$	52.4 ± 0.19	0.391	0.627	2.620	$\mathbf{0.33} \pm 0.19\%$	1.2%

Table 2: Comparing of our approach, VCP, to other methods and T5 baseline on ViGGO dataset. SLSER represents how many percentage of the sentences contains slot error. We mainly compare SER and BLEU. The subscript of each method represents the base model the method is using.

Model	BLEU	MET.	ROUGE.	SER ↓	SLSER
T5-small$_{greedysearch}$ baseline	67.0	0.454	0.692	1.60%	9.9%
T5-small$_{beamsearch}$ baseline	66.7	0.453	0.694	2.85%	11.6%
T5-base$_{greedysearch}$ baseline	66.8	0.459	2.282	1.85%	-
T5-base$_{beamsearch}$ baseline	66.7	0.453	0.697	3.94%	-
S2S(Juraska et al., 2019)	66.2	0.445	0.677	0.91%	-
K&M(Kedzie and McKeown, 2020)	66.3	0.453	0.693	**0**	-
SEA-GUIDE$_{T5-small}$ (Juraska and Walker, 2021)	67.5	0.453	0.690	0.04%	0.25%
SEA-GUIDE$_{T5-base}$(Juraska and Walker, 2021)	68.2	0.454	0.691	0.05%	0.32%
DM-NLG no postprocess(Seifossadat and Sameti, 2023)	66.7	0.456	0.691	0.03%	-
DM-NLG postprocess: GPT-2(Seifossadat and Sameti, 2023)	68.6	0.482	0.713	0.03%	-
VCP$_{T5-small}$	67.0 ± 0.18	0.451	0.690	**0.002%**	0.015%

Table 3: Comparison of our method VCP to T5 baseline and other methods on E2E dataset. SLSER represents how many percentage of the sentences contains slot error. We mainly compare SER and BLEU. The subscript of each method represents the base model the method is using.

script[1]. For semantic error evaluation, we use the SER which measures the error rate of slot values in the generated text. We apply the same auto slot evaluation script used in the SEA-GUIDE project[2], encompassing hundreds of evaluation rules. The auto evaluation script exhibits a 94% agreement rate aligning with human judgment.

The Slot Error Rate (SER) is calculated by dividing the number of slot errors by the total number of slots. For instance, if a sentence contains 1 slot error but has a total of 8 slots, the slot error rate for this incorrect sentence would be 12.5% rather than 100%. People unfamiliar with SER might underestimate the severity of a low SER rate. Therefore, we introduce the Sentence-Level Semantic Error Rate (SLSER), which counts the percentage of sentences with semantic errors. This analysis was conducted manually to highlight the severity of the slot errors. We primarily use SER to measure slot errors, as this is the common metric used in other research.

[1]https://github.com/tuetschek/e2e-metrics
[2]https://github.com/jjuraska/data2text-nlg

4.2 Setup

We observed a substantial variance in the SER of the T5 baseline, SEA-GUIDE, and our method when applied to the ViGGO dataset. To ensure a fair comparison, we trained 5 instances each of the T5-small and T5-base models, each for 20 epochs. We also ran SEA-GUIDE and our VCP 5 times for T5-base and T5-small, calculating the mean and variance. All optimized models were selected based on validation loss. In the VCP project, we used a batch size of 10 and a maximum sentence length of 300 tokens. All tests were conducted on a single RTX 3090 GPU, with a linear learning rate scheduler. More comprehensive information regarding the prompt training hyperparameters is provided in the Appendix. For the E2E datasets, we followed the same procedure. The experimental results for other methods, where standard deviations are not reported, are as presented in Tables 2 and 3, and have been taken directly from the original papers.

	BLEU	SER
T5-small$_{beamsearch}$ baseline	53.2	0.89%
VCP$_{T5small}$	52.6	**0.41%**
GPT3.5 examples	25.1	7.3%
GPT3.5 selected examples	22.3	6.72%
GPT4 examples	30.7	0.90%
GPT4 selected examples	27.4	**0**

Table 4: Performance comparison between large language models, T5 baseline and our method

	BLEU	SER
T5-small$_{beamsearch}$ baseline	53.2	0.89%
VCP$_{T5small}$	52.6	0.41%
Remove position information	52.4	0.72%
Fine-tune T5(not use prompt)	51.4	0.70%
No prompt initilization	52.6	0.65%
Directly sampling the output	50.6	0.62%

Table 5: Ablation study on ViGGO dataset

4.3 Performance Comparison

As demonstrated in Table 2, we contrast our methodology, VCP, with the T5 baseline and other methods using the ViGGO dataset. Our method exhibits a notable advantage in terms of reducing the Slot Error Rate (SER) while maintaining comparable non-SER scores to the T5 baseline. The SER result reported by K&M cannot be directly compared with our method, given that we employ different methodologies for calculating SER. Our VCP method reduces SER from 0.89% to 0.41% on T5-small and from 0.60% to 0.33% on T5-base.

As shown in Table 3, for the E2E dataset, our VCP method not only retains text generation quality on non-SER evaluation metrics but also reduces SER from over 2.5% for the T5 beam search baseline to almost 0. This is lower than other methods, except for K&M. Although K&M performs well on the E2E dataset, it struggles to maintain text generation quality, achieving a 48.5 BLEU score on the ViGGO dataset. DM-NLG (Seifossadat and Sameti, 2023), with post-processing using GPT-2, reduces SER to 0.03% and improves text fluency on non-SER evaluation metrics. However, their method incorporates a post-processing stage that employs a significantly larger language model, GPT-2, to enhance the fluency of the initial prediction. This makes the text quality comparison with our method somewhat unfair. Our method achieves a higher BLEU score compared to DM-NLG without post-processing, while reaching a lower SER score.

4.4 Comparing to LLMs

The findings are summarized in Table 4, where we evaluate GPT3.5 and GPT4. It is critical to note that the performance of GPT3.5, when given five in-context examples of the input intent type (as shown in Appendix-Prompt with One Example for Each Intent), significantly underperforms compared to the T5 baseline in terms of the BLEU score. This underperformance is also evident when GPT3.5 is given one in-context example for every intent from the ViGGO dataset (details in Appendix-Prompt with Selected Examples). The same prompting strategy was applied to GPT4. Although GPT4 does not achieve a high BLEU score, it attains considerably lower SER scores. Remarkably, GPT4 achieved a 0% SER with selected in-context examples, showcasing its exceptional capability to accurately follow instructions.

We believe the lower prediction quality generated by GPT3.5 and GPT4 is primarily due to the complexities involved in expressing the relationship between the input and output through in-context examples and prompts. For instance, despite being provided with five distinct request attribute examples and clear prompt explanations (as detailed in Appendix-Prompt with Selected Examples), GPT3.5 falls short in accurately replicating the desired tone and often misconstrues the intended meaning of the request attribute intent in the input data. For example, in the data shown in Appendix-Prompt with Selected Examples, the intent of the request attribute suggests that the user is seeking to ascertain whether their feelings are average. GPT3.5 misinterprets this, inferring that the input data is attempting to verify all available information. There are myriad ways to interpret how an AI model should convert input data into text. However, the true relationship can be more effectively understood through training a language model on thousands of examples, rather than presenting it with a limited number of in-context examples and descriptions. Consequently, supervised training continues to play an essential role in data-to-text generation models.

5 Ablation Study

In Table 5, we conduct an ablation study using the T5-small model on the ViGGO dataset.

5.1 Removing Position Information

In this experiment, we evaluate the importance of placing error-correcting prompts adjacent to the slot errors. To this end, we remove the slot error position information by always positioning the error-correcting prompts at the front of the inputs. Table 5 illustrates that slot error rates rise when we remove information regarding error locations, underscoring the advantage of highlighting the probable sites of errors.

5.2 Fine-tuning the Entire Model (T5 no prompt)

In this experiment, we assess whether using error-correcting prompts results in better performance than fine-tuning the entire model on the generated training data. After training a T5 model on the training dataset, we further fine-tune it with a learning rate of 5ê-3 for 10 epochs using the prompt training dataset (with error-correcting prompts removed) created in the Data Generation section. We compare its performance to VCP, which only trains error-correcting prompts. As shown in Table 5, the BLEU score and SER generated by 'Fine-tune T5' are noticeably lower than those achieved by prompt tuning methods, demonstrating the importance of using prompts. The reason lies in the fact that prompts not only label the error locations but also allow the original model to remain frozen. The original model, trained on ground-truth data labeled by humans (unlike ground-truth in the prompt training datasets created by T5 models which may contain errors), excels at producing high-quality texts. Utilizing prompts enables the minimally affected well-trained T5 model, thereby yielding better-quality text outputs (high BLEU score) and learning a more generalizable ability to guide language models in reducing slot errors.

5.3 Remove the prompt initialization training process

During the training process, we first train prompt initialization and then fine-tune the initialization embedding, as opposed to directly fine-tuning a randomly initialized embedding. In this experiment, we aim to evaluate the significance of prompt initialization by comparing the performance of VCP before and after using the prompt initialization step (Not initial.). As shown in Table 5, there is a decline in the BLEU score and an increase in the SER score after the removal of prompt initialization, thereby emphasizing its vital role in maintaining the quality of text generation.

5.4 Sampling the best prediction directly

In our project, we use error-correcting prompts to guide fine-tuned T5 models in correcting their slot errors. We compare our method to directly sampling 10 outputs and selecting the best prediction using an SER score. The results demonstrate that while the direct sampling method reduces the SER, the quality of the generated texts diminishes compared to the prompt-based method.

The main reason for this is that the slot error checker does not always accurately recognize when generated texts use different words to mention slot information. This can result in the original prediction being incorrectly identified as having errors, leading to the selection of alternative predictions which may have lower text fluency. Error-correcting prompts, on the other hand, can learn generalizable knowledge during training, allowing them to guide the T5 model beyond the sampling search range and perform more natural predictions.

6 Discussion and Conclusion

By utilizing a feedback system pipeline, our method achieves the lowest SER compared to other methods, while still maintaining a comparable level of text generation quality.

Our approach attains a lower SER and maintains text quality primarily because it does not overly rely on predefined rules, which can be inaccurate in complex scenarios. Our specialized training method enables accurate regeneration even with imprecise feedback. Additionally, our feedback system not only informs the model about the correctness of its output but also indicates where the errors are located.

Our method is particularly suitable for smaller models that are incapable of reasoning based on natural language feedback. To preserve the efficiency advantage of such models, we use a basic verifier. While this verifier is not highly accurate, it is both easy to implement and fast, making it an efficient choice.

We hope VCP can be adapted for other applications that require a feedback system, especially in scenarios where providing accurate feedback or understanding feedback is challenging. Potential applications include text-to-image generation, story summarization, and text-to-SQL generation.

7 Limitations

Our method effectively reduces slot errors; however, it also slightly decreases text fluency. This decrease in fluency occurs because we train the prompt tokens using ground truth data generated by the fine-tuned language model itself, which may sometimes be inaccurate or sound unnatural. To improve text fluency, the introduction of a filter to eliminate low-quality text or the use of post-processing tools might be beneficial.

Additionally, it's important to note that our approach has been specifically tested on data-to-text generation tasks. We have not yet explored the potential of applying this method to other types of tasks. Future research may investigate its applicability and effectiveness in different domains or for various natural language processing challenges.

References

Vidhisha Balachandran, Hannaneh Hajishirzi, William Cohen, and Yulia Tsvetkov. 2022. Correcting diverse factual errors in abstractive summarization via post-editing and language model infilling. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9818–9830, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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*, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Shailza Jolly, Zi Xuan Zhang, Andreas Dengel, and Lili Mou. 2022. Search and learn: improving semantic coverage for data-to-text generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10858–10866.

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*, Tokyo, Japan. Association for Computational Linguistics.

Juraj Juraska and Marilyn Walker. 2021. Attention is indeed all you need: Semantically attention-guided decoding for data-to-text NLG. In *Proceedings of the 14th International Conference on Natural Language Generation*, pages 416–431, Aberdeen, Scotland, UK. Association for Computational Linguistics.

Mihir Kale and Abhinav Rastogi. 2020. Template guided text generation for task-oriented dialogue. *arXiv preprint arXiv:2004.15006*.

Zdeněk Kasner and Ondrej Dusek. 2022. Neural pipeline for zero-shot data-to-text generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Dublin, Ireland. 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, Online. 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, Prague, Czech Republic. Association for Computational Linguistics.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 61–68, Dublin, Ireland. Association for Computational Linguistics.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.

Sanket Vaibhav Mehta, Jinfeng Rao, Yi Tay, Mihir Kale, Ankur Parikh, and Emma Strubell. 2022. Improving compositional generalization with self-training for data-to-text generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Dublin, Ireland. Association for Computational Linguistics.

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, Saarbrücken, Germany. Association for Computational Linguistics.

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.

Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, et al. 2023. Check your facts and try again: Improving large language models with external knowledge and automated feedback. *arXiv preprint arXiv:2302.12813*.

Ratish Puduppully, Li Dong, and Mirella Lapata. 2019. Data-to-text generation with content selection and planning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 6908–6915.

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. *J. Mach. Learn. Res.*, 21(1).

Clément Rebuffel, Laure Soulier, Geoffrey Scoutheeten, and Patrick Gallinari. 2019. A hierarchical model for data-to-text generation.

Elham Seifossadat and Hossein Sameti. 2023. Improving semantic coverage of data-to-text generation model using dynamic memory networks. *Natural Language Engineering*, pages 1–26.

Kumar Shridhar, Harsh Jhamtani, Hao Fang, Benjamin Van Durme, Jason Eisner, and Patrick Xia. 2023a. Screws: A modular framework for reasoning with revisions. *arXiv preprint arXiv:2309.13075*.

Kumar Shridhar, Koustuv Sinha, Andrew Cohen, Tianlu Wang, Ping Yu, Ram Pasunuru, Mrinmaya Sachan, Jason Weston, and Asli Celikyilmaz. 2023b. The art of llm refinement: Ask, refine, and trust. *arXiv preprint arXiv:2311.07961*.

Yixuan Su, David Vandyke, Sihui Wang, Yimai Fang, and Nigel Collier. 2021. Plan-then-generate: Controlled data-to-text generation via planning. *arXiv preprint arXiv:2108.13740*.

Ramakrishna Vedantam, C. Lawrence Zitnick, and Devi Parikh. 2015. Cider: Consensus-based image description evaluation.

Xinnuo Xu, Ondřej Dušek, Verena Rieser, and Ioannis Konstas. 2021. AggGen: Ordering and aggregating while generating. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Online. Association for Computational Linguistics.

Tianci Xue, Ziqi Wang, Zhenhailong Wang, Chi Han, Pengfei Yu, and Heng Ji. 2023. Rcot: Detecting and rectifying factual inconsistency in reasoning by reversing chain-of-thought. *arXiv preprint arXiv:2305.11499*.

Xunjian Yin and Xiaojun Wan. 2022. How do Seq2Seq models perform on end-to-end data-to-text generation? In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7701–7710, Dublin, Ireland. Association for Computational Linguistics.

A Prompt with Selected Examples

We demonstrate an example of the prompt we use for GPT3.5. We demonstrate the prompt for random select the example from training dataset with the intent that is the same as the intent in the test example (request attribute).

PROMPT:

please perform data-to-text generation for me. Domain is video game. The words before the bracket are intentions.

For example, when the intention is give opinion, then the output should be a sentence that asks for opinion.

when the intention is verify attribute, then the output should be a sentence that try to verify the attribute.

Example: Input: request attribute(esrb[]) Output: Are there any ESRB content ratings which you give preference to when picking a game to play?

Input: request attribute(release year[]) Output: Can you think of a year, in which video games were particularly good?

Input: request attribute(esrb[]) Output: Are there any ESRB content ratings which you give preference to when picking a game to play?

Input: request attribute(esrb[]) Output: Are there any ESRB content ratings which you give preference to when picking a game to play?

Input: request attribute(developer[]) Output: Which game developer do you think is the best?

Question: Input: verify attribute(name[little big adventure], rating[average], has multiplayer[no], platforms[playstation])

Output:

Answer by GPT3.5: Can you confirm that Little Big Adventure has an average rating and does not have multiplayer? Also, is it available on PlayStation?

GroundTruth: ['I remember you saying you found Little Big Adventure to be average. Are you not usually that into single-player games on PlayStation?', "Earlier, you stated that you didn't have strong feelings about PlayStation's Little Big Adventure. Is your opinion true for all games which don't have multiplayer?", 'I recall that you were not that fond of Little Big Adventure. Does single-player gaming on the PlayStation quickly get boring for you?']

B Prompt with One Example for Each Intent

We demonstrate an example of the prompt we use for GPT3.5. In the example in the Appendix-Prompt with One Example for Each Intent, we demonstrate the prompt we use for all of the test cases. The prompt include one example for each intent.

PROMPT:

please perform data-to-text generation for me. Domain is video game. the words before the bracket are intentions. For example, when the intention is give opinion, then the output should be a sentence that asks for opinion. when the intention is verify attribute, then the output should be a sentence that try to verify the attribute.

Example for each intention: Input: give opinion(NAME [SpellForce 3], RATING [poor], GENRES [real-time strategy, role-playing], PLAYER PERSPECTIVE [bird view]) Output: I think that SpellForce 3 is one of the worst games I've ever played. Trying to combine the real-time strategy and role-playing genres just doesn't work, and the bird's eye view makes it near impossible to play.

Input: verify attribute(NAME [Little Big Adventure], RATING [average], HAS MULTIPLAYER [no], PLATFORMS [PlayStation]) Output: I recall that you were not that fond of Little Big Adventure. Does single-player gaming on the PlayStation quickly get boring for you?

Input: confirm(NAME [Hellblade: Senua's Sacrifice], RELEASE YEAR [2017], DEVELOPER [Ninja Theory]) Output: Oh, do you mean the 2017 game from Ninja Theory, Hellblade: Senua's Sacrifice?

Input: request(SPECIFIER [interesting]) Output: Have you played any interesting games lately?

Input: suggest(NAME [Half-Life 2], GENRES [shooter], PLAYER PERSPECTIVE [first person]) Output: Do you also enjoy playing first-person shooters, such as Half-Life 2?

Input: request explanation(RATING [poor], HAS MAC RELEASE [yes]) Output: What is it about Mac games that you find so disappointing?

Input: inform(NAME [Max Payne 3], RELEASE YEAR [2012], GENRES [action-adventure; shooter], MULTIPLAYER [yes]) Output: Max Payne 3 is a multiplayer action-adventure shooter from 2012.

Input: request attribute(AVAILABLE ON STEAM []) Output: Do you prefer playing games that you can get on Steam?

Question: Input: YOUR INPUT QUESTION Output:

Model	initial. lr	train. lr	epochs	prompt token num
VCP$_{T5-small}$	0.01	0.005	5	6
VCP$_{T5-base}$	0.01	0.01	2	3

Table 6: The following details pertain to the training process of our VCP method for experiments on the ViGGO datasets. 'Initial. lr' stands for the initial learning rate used for prompt initialization, while 'Train. lr' represents the learning rate used for prompt tuning. 'Epochs' refers to the number of epochs for prompt tuning.

Model	initial. lr	train. lr	epochs	prompt token num
VCP$_{T5-small}$	0.01	0.01	10	6

Table 7: The following details pertain to the training process of our VCP method for experiments on the E2E datasets. 'Initial. lr' stands for the initial learning rate used for prompt initialization, while 'Train. lr' represents the learning rate used for prompt tuning. 'Epochs' refers to the number of epochs for prompt tuning.

C Slot Error Checking Examples

For instance, *has_linux_released[yes]*, the first step is to see if *linux* appears in the prediction. If it does not, error-correcting prompts are positioned beside the slot value. If it does appear, we employ simple dependency parsing rules and POS tags to ascertain if any negation words are linked to *linux*. If negation words are found, the slot value is marked with a error-correcting prompts. If no negation words are present, we infer that there are no slot errors concerning *has_linux_released[yes]*. Conversely, if the slot value is *no*, such as in *has_linux_released[no]*, the initial step is to check for the mention of *linux* in the prediction. If *linux* is not mentioned, it is assumed that no slot errors exist. However, if *linux* is mentioned, we look for any associated negation words. If none are found, error-correcting prompts are placed beside the *no* slot value, indicating a potential slot error. If negation words are present, we presume the absence of slot errors.

D Training parameters

We use the learning rate begins at 0.01 and reduces gradually over 20 epochs for prompt embedding initialization. We use 3 error-correcting prompts for T5-base and 6 error-correcting prompts for T5-small. More details can be seen in Table 6 and 7.

E Experiment details

On the ViGGO dataset, we run the T5 baseline, SEA-GUIDE project, and our VCP for 5 times. We report the mean and variance in Table 2 and reproduce the experiment results of S2S, DT, and K&M from the SEA-GUIDE paper. We also run the ablation study once and report the results in Tables 4 and 5.

For the E2E dataset, we run our VCP and reported the mean and variance. We report the experiment results from the DM-NLG paper and the SEA-GUIDE paper in Table 3.

We use the SER auto-evaluation script from the SEA-GUIDE Github project to evaluate SER on both the ViGGO and E2E datasets. However, when evaluating the model on the E2E dataset, the SER evaluation script is not accurate. As a result, we manually check every prediction labeled as having slot errors by the SER evaluation script. When applying GPT3.5 and GPT4 to the ViGGO dataset, we also manually check every prediction labeled as incorrect by the SER evaluation script.

F Prompt Initialization Details

To achieve this, we train error-correction prompts such that inserting them does not alter the output of the fine-tuned T5 model. Specifically, we remove the prompts from the unseen prompted input, then forward these inputs to the fine-tuned T5 model for prediction. We use these predictions as our training targets and the unseen prompted inputs as training input. We train the error correction prompt tokens while remain fine-tuned T5 model frozen. Performing prompt tuning on such a initialized prompt instead of the random initialized prompt is demonstrated to have better performance as shown in the prompt initialization ablation study.