Large-Scale Constraint Generation Can LLMs Parse Hundreds of Constraints?

Anonymous ACL submission

Abstract

Recent research has explored the *constrained generation* capabilities of Large Language Models (LLMs) when explicitly prompted by few task-specific requirements. In contrast, we introduce *Large-Scale Constraint Generation (LSCG)*, a new problem that evaluates whether LLMs can parse a large, fine-grained, generic list of constraints. To examine the LLMs' ability to handle an increasing number constraints, we create a practical instance of LSCG, called *Words Checker*. In Words Checker, we evaluate the impact of model characteristics (e.g., size, family) and steering techniques (e.g., *Simple Prompt*, *Chain of Thought*, *Best of N*) on performance. In addition, we propose *FoCusNet*, a small and dedicated model that parses the original list of constraints into a smaller subset to help the LLM focus on relevant constraints. Experiments reveal that existing solutions suffer a significant performance drop as the number of constraints increases, with FoCusNet showing at least an 8-13% accuracy boost.

1 Introduction

Instructions are prompts or directives, written in natural language, that guide the model to perform a specific task (Ouyang et al., 2022). The recent literature has extensively studied the ability of Large Language Models (LLMs) to follow instructions requiring complex reasoning (Wang et al., 2023), focusing on multiple requirements and for multiple rounds (He et al., 2024c,b), and even dealing with long texts (Bai et al., 2024; Li et al., 2024). To address the real-world urgency for controllable outputs (Liu et al., 2024a; Hassan et al., 2024), researchers have also investigated whether LLMs, provided with clear indications of the expected answer, could support *constrained generation* (e.g., "the answer must contain exactly N words") (Sun et al., 2023; Yao et al., 2024; Xia et al., 2024).

In this paper, we take a step further in defining instruction-following tasks. In particular, instead

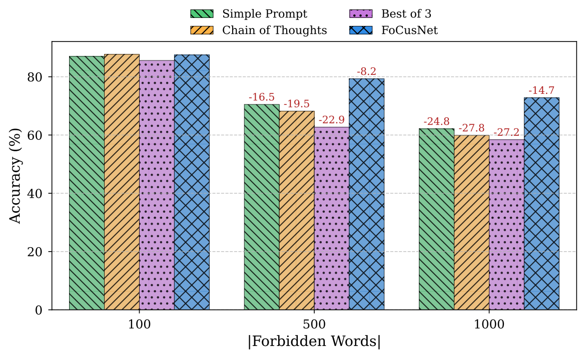

Figure 1: FoCusNet significantly outperforms typical LLM inference methods on the proposed Words Checker task (*DeepSeek-R1-Distill-Llama-8B*). Red numbers indicate differences compared to 100-word scenario.

of the few task-specific indications the literature has used so far, we focus on scenarios with a high number of fine-grained but general constraints that the model must respect to generate a valid answer. Consider the example in Fig. 2. The model faces a social task (e.g., "be a good visitor in an Islamic country"), and can access a comprehensive travel guide with generic information on how to achieve the goal (i.e., long list of constraints). Could the LLM, with the sole aid of the generic travel guide and no other explicit instruction, realise that "inviting a Muslim for a beer after prayer" (Naous et al., 2024) is not a good way to solve the task?

We call this new framework *Large-Scale Constraint Generation (LSCG)*. LSCG examines whether LLMs can replicate humans' *practical intelligence* (Sternberg, 1986), i.e., the ability to interpret and adapt to the context. In particular, facing LSCG the model is not tasked to solve complex reasoning problems, but rather i) to consult broad and generic guidelines (e.g., travel guide, but also updated documentation while coding (Wang et al., 2024; Deng et al., 2024)), ii) to identify the requirements relevant for the specific problem, and iii) to apply them to derive a valid solution.

As it is currently unclear whether and how LLMs' capabilities could scale with the hundreds

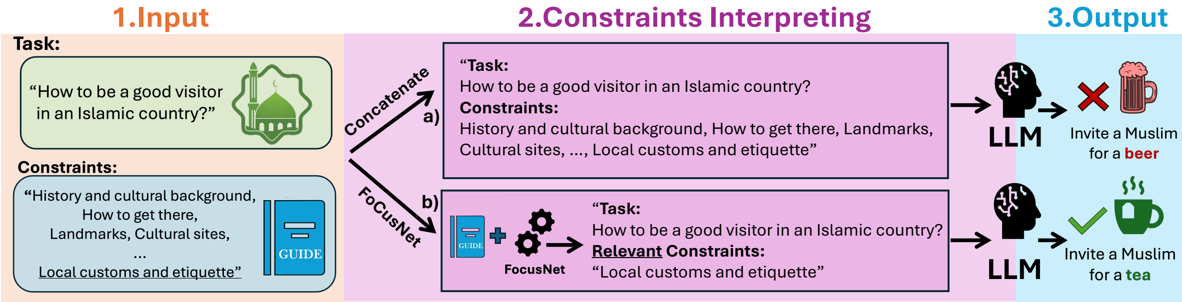

Figure 2: In LSCG, the model must generate a **valid answer** while adhering to an **input task** and a **long list of constraints**. In the example, this can be done either by (a) directly interpreting the **concatenated** task and constraints or (b) using a **FoCusNet** to **extract relevant constraints**. The first approach may lead to **inappropriate responses** (e.g., offering beer to a Muslim (Naous et al., 2024)), while the second ensures **valid answers**.

(if not thousands) of constraints that a travel guide or some code documentation could provide, we implement a concrete instance of LSCG, *Words Checker*. We design Words Checker as a simple problem, not requiring particular reasoning skills, to explicitly study how the performance of LLMs while solving the task is affected by the number of constraints. In Words Checker, the model is given as input a list of forbidden words and a sample sentence. The task is to classify the sentence as *valid* (i.e., does not contain forbidden words) or *invalid* (i.e., contains at least one forbidden word).

We create different instances of Words Checker with increasingly larger lists of forbidden words (e.g., 100, 500 and 100). Then, we systematically evaluate how features such as model family – Meta's *LLama* (Grattafiori et al., 2024) vs. Deepseek's *R1* (DeepSeek-AI et al., 2025)), size – 8B vs. 70B, and Test Steering Strategies (TSS)– *Simple Prompt*, *Chain of Thought* (Wei et al., 2022b; Lightman et al., 2024) and *Best of N* (Chen et al., 2024b; Madaan et al., 2023) affect the results.

Furthermore, inspired by *Retrieval Augmented Generation (RAG)* (Lewis et al., 2020) and the recent literature (Cobbe et al., 2021; Shi et al., 2024), we propose *FoCusNet (Focused Constraints Net)*, a lightweight and customizable model to parse the originally large list of constraints into a smaller set of constraints relevant to the task, helping the LLM to better focus. In Words Checker, FoCusNet is a $\sim$ 300k parameters model that we train to determine whether a set of words is present in a sentence. During inference, it preprocesses the long list of forbidden words and parses it into a smaller set of potential suspects, allowing the LLM to focus more effectively on meaningful instances.

The results of a distilled 8B LLM in Words Checker, shown in Fig. 1, are striking: traditional Test Steering Strategies, including simple prompting, suffer a drastic performance drop – down to $\sim$ 27.8% accuracy. Manual analysis reveals that the model often processes words individually, losing focus, and sometimes conflating its reasoning process with the actual task. For example, it may incorrectly assert that a word is present simply because it appears in a self-generated list. Our approach proves the most robust, leveraging the synergy between two models. FoCusNet, trained to detect the presence of words with accuracy 90%, effectively narrows the search space (i.e., average of 30 suspicious words out of 1000). The LLM, in turn, benefits from this reduced scope, filtering out false positives from FoCusNet and improving overall accuracy. In sum, our contributions are:

– **Large-Scale Constraint Generation**: A novel problem to evaluate the ability of current LLMs to automatically parse a large number of constraints and identify the relevant ones.

– **Words Checker**: A practical example of LSCG where the model identifies invalid sentences as the number of forbidden words increases. We systematically experiment 2 models (*LLama* and *R1*), 2 model sizes (8B and 70B), and 3 TSS (prompt-based, CoT, Best of N).

– **FoCusNet**: A small dedicated model that works in conjunction with the LLM, helping it to better focus on relevant constraints.

– **Code and Datasets**: To reproduce Words Checker and FoCusNet and help the community benchmarking LSCG.

2 Related Work

Instruction-Following abilities of LLMs. The challenge of constraining textual generation has been studied since the early days of NLP (Hu et al., 2017), but the rise of LLMs has dramatically increased expectations beyond merely "producing plausible text" (Brown et al., 2020; Wei et al., 2022a). Modern LLMs are expected to follow complex instructions, handle multiple constraints across interactions (He et al., 2024c,b), and process long texts (Bai et al., 2024; Li et al., 2024). Yet, this problem remains unsolved. Studies show that LLMs struggle with adherence to rules (Mu et al., 2024), format following varies widely across domains (Xia et al., 2024), open-source models are still behind closed source solutions (Wang et al., 2023) and smaller models still perform poorly in structured tasks (Wang et al., 2025). Most of the previous evaluations assume interactive chat-like settings, with few clear user instructions specific to the required task. In contrast, we contribute to this line of research by examining how LLMs perform when given an extensive list of fine-grained yet generic requirements to satisfy.

Instruction Tuning. Given these challenges, instruction tuning might seem like a natural candidate for improving adherence to complex and fine-grained constraints. Prior work has highlighted its role in enhancing generalization capabilities (Chung et al., 2022; Mishra et al., 2022; Thoppilan et al., 2022), and even a small set of high-quality instructions can lead to performance gains (Zhou et al., 2023; Chen et al., 2024a). However, despite well-established guidelines for crafting such instructions (Zhao et al., 2024; He et al., 2024a; Zhang et al., 2024), instruction tuning remains costly and resource-intensive. This makes it unsuitable for large-scale applications that require customization (Chang et al., 2016; Zhang and Chen, 2020), continuous knowledge updates (Lewis et al., 2020), or, like our example in Fig. 1, cultural adaptation (Adilazuarda et al., 2024; Kotek et al., 2023). Instead, we argue that LLMs should, like humans, handle unfamiliar constraints by leveraging external knowledge sources while relying on their reasoning abilities to interpret and respond accordingly. Consequently, we do not employ instruction tuning to further specialize our models.

Test Steering Strategies. Rather than modifying a model through instruction tuning, an alternative approach is to guide LLM outputs at inference using test-time steering strategies. These methods enhance rule adherence without the cost and inflexibility of fine-tuning. Prior research has explored various controlled generation techniques to enforce constraints (Hu et al., 2018). LLMs have shown strong performance with simple interventions like Chain-of-Thought (CoT) prompting (Wei et al., 2023). However, studies suggest that such methods alone may be insufficient for handling fine-grained, hard constraints (Sun et al., 2023). To address this, researchers have investigated best-of-K selection (Nakano et al., 2022; Stiennon et al., 2020), where multiple independent samples are generated, scored, and ranked to select the most suitable output. Other approaches include rejection-sampling-based methods (Liu et al., 2024b), reward-model-guided decoding (Yang and Klein, 2021; Deng and Raffel, 2023), and constraint-aware streaming algorithms (Krause et al., 2021; Liu et al., 2021). Building on this body of work, we assess the rule-following capabilities of LLMs using various test-time steering strategies.

Auxiliary Modules for LLMs. In this paper, we present FoCusNet, a modular support model that enhances LLMs' ability to follow constraints. Unlike base model modifications, FoCusNet acts as an auxiliary module that identifies and prioritizes relevant constraints, guiding the LLM's generation process. It provides an intermediate solution between resource-heavy instruction tuning and simpler test-time steering methods, which, while more efficient, may struggle with complex tasks.

Similar approaches using specialized support models for LLMs have been explored in various text generation tasks. For example, retrieval-augmented generation (RAG) (Lewis et al., 2020; Shi et al., 2024) improves LLM responses by incorporating external knowledge, while classifier-based safeguards promote responsible generation (Sharma et al., 2025). Furthermore, researchers have also developed classifier-based content moderation systems (Chi et al., 2024; Inan et al., 2023; Rebedea et al., 2023) and output filtering techniques to address jailbreak vulnerabilities (Kim et al., 2024),

3 Large-Scale Constraint Generation

In this Section, we formally define LSCG, relate Test Steering Strategies techniques with LSCG and finally introduce FoCusNet.

3.1 Formal Definition

In constrained generation, LLMs autoregressively generate an output sequence y according to an input task t and a set of constraints $c = \{c_1, c_2, \ldots, c_C\}$. LSCG is a specific case of constrained generation characterized by a large number of constraints (i.e., $C \geq 100$). We suppose both t, and the constraints c_i with $i \in C$ to be string-based. Although this

Table 1: Summary of how different steering solutions produces the final query $q = e(t) \parallel p(c)$.

Test steering	**Enhance** - $e(t)$	**Parse** - $p(c)$
Simple Prompt	t	$c_1 \parallel c_2 \parallel \cdots \parallel c_C$
Chain of Though	$t \parallel g$	$c_1 \parallel c_2 \parallel \cdots \parallel c_C$
Best of N	$t \parallel g$	$y_1 \parallel y_2 \parallel \cdots \parallel y_N$
FoCusNet	$t \parallel g$	$f_\phi(c)$

assumption does not cover the most general case (see Sect. 6), it is sufficient to model real-world scenarios such as the travel guide and documentation examples of Sect. 1.

We define the LLM input *query*: $q = e(t) \parallel p(c)$, where $\parallel$ is the concatenation. Specifically, here e and p are *Test Steering Strategies* that can be applied to improve model performance: e is a function that *enhance* the definition of the task, while p helps *parsing* the constraints. We provide more details in the next section.

We represent the LLM as a function $f_\theta : q \rightarrow y$. This means that the LLM generates an answer y as $y = f_\theta(q)$ according to its pre-trained weights θ. A model-generated answer y is valid for a given query q if it correctly solves the task t while adhering to the constraints c.

3.2 Existing Test Steering Strategies

Here, we list the most prominent TSS previously identified in the literature and examine how they apply in our formulation. We provide a summary in Tab 1.

Simple Prompt. As both t and c are text-based, a natural approach is to simply *concatenate* them: $q = t \parallel c_1 \parallel c_2 \parallel \cdots \parallel c_C$.

Chain of Thought (CoT). To enhance the reasoning capabilities of the LLM, we modify t by appending a guide phrase g, such as "*Think step by step*": $q = t \parallel g \parallel c_1 \parallel c_2 \parallel \cdots \parallel c_C$.

Best of N. Finally, to improve the interpretation of the C constraints, we can involve a panel of N judges (e.g., independent runs of the model), each performing CoT reasoning independently, followed by a recap step to produce the final answer. Formally, let $y_n = f_{n,\theta}(t \parallel g \parallel c_1 \parallel c_2 \parallel \cdots \parallel c_C)$ denote the answer of the nth judge, where $n \in N$. Then, we can aggregate all the responses into a refined query: $q = t \parallel y_1 \parallel y_2 \parallel \cdots \parallel y_N$.

3.3 FoCusNet

Definition. Here, the goal is to learn an approximation of $p(c) : c \rightarrow k$ to reduce the large set of C constraints c to a more compact subset $k \in K$ of relevant constraints. To do that, we introduce a dedicated model, FoCusNet. Specifically, we define FoCusNet as a function f_ϕ with learnable parameters ϕ, trained on task-specific data to filter relevant constraints. Once trained, FoCusNet applies this filtering as $k = f_\phi(c)$, which yields the final query formulation: $q = t \parallel g \parallel k \parallel$.

Training FoCusNet. We train FoCusNet to perform a binary classification task over individual constraints. Specifically, FoCusNet operates on triplets $(\hat{c}, s, l)$. Here, $\hat{c} = \{c_1, c_2, \ldots, c_M\}$ is a subset of M constraints from c; s is a text-based instance where the constraint is satisfied or violated, and $l \in \{0, 1\}$ is a label indicating whether the constraint is violated (1) or not (0). For example, consider Fig. 2. The set of constraints is $\{c_1 =$"*Respect local customs and etiquette when visiting an Islamic country*"$\}$; the instance is $s =$ "*Invite a Muslim for a beer*"; the corresponding label l is violated ($l = 1$).

Inference with FoCusNet. During inference, FoCusNet receives as input the tuple of constraints and task (c, t) and generates a *relevance mask*, $m = \{m_1, m_2, \ldots, m_C\}$ with $m_i \in \{0, 1\}$ and $i \in C$. The mask determines which constraints are relevant for the task. Applying the mask yields the reduced set: $k = \{c_i \mid m_i = 1, \forall i \in C\}$.

As in any alerting system, FoCusNet aims at compromising *recall* and *precision*. Ideally, we would like FoCusNet to reduce the number of false positives, i.e., irrelevant constraints mistakenly included. In fact, a large number of false positives leads to a larger and noisy set k. At the same time, it is essential to minimize false negatives, as excluding relevant constraints could hinder the LLM's ability to generate valid outputs.

4 Methodology

In this section, we discuss the engineering of Words Checker and, consequently, FoCusNet's training.

4.1 Words Checker

Problem Definition. Words Checker is an instance of LSCG, where an LLM must classify a sentence as *valid* or *invalid* based on a dynamically provided list of forbidden words. Formally, given a sentence $S = (w_1, w_2, \ldots, w_n)$ and a set of forbidden words $F = \{w_{f1}, w_{f2}, \ldots, w_{fm}\}$, the model must determine whether S contains any word morphologically related to an element of F. A sentence is classified as *invalid* if $\exists w_{fi} \in F$ such that w_{fi} is a root or morphological variant of any $w_j \in S$,

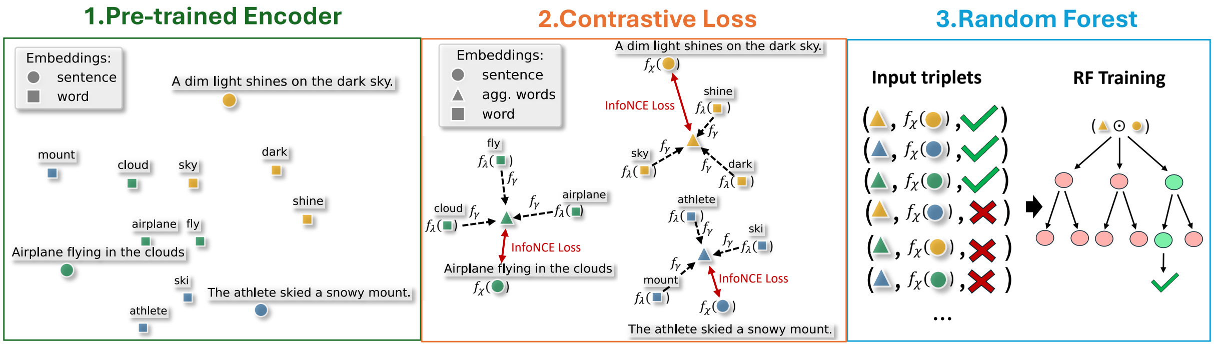

Figure 3: Training pipeline of FoCusNet for Words Checker. The model receives as input a batch of sentences and words. In Phase 1, FoCusNet uses a **frozen pre-trained model** to map the input into sentences (circles) and words (squares) embeddings. Then, in Phase 2, FoCusNet learns to **refine the sentence embeddings** (f_χ) and to **aggregate the words embeddings** (f_γ, f_λ) with a InfoNCE contrastive loss. Eventually, in Phase 3 FoCusNet train a Random Forest to **discriminate positive and negative examples**.

and *valid* otherwise. For example, given the sentence "The athlete skied a snowy mountain" and $F = \{\text{ski}\}$, the output should be *invalid*, since "skied" is a morphological variant of "ski". In contrast, for "The bathroom has recently been cleaned" and $F = \{\text{restroom}\}$, the output should be *valid*, as no word in S morphologically relates to "restroom".

Ratio behind Words Checker. We explicitly design Words Checker to study the impact of an increasing number of forbidden words on LLM performance. Therefore, unlike other constrained generation problems, this task does not require complex reasoning. Instead, we engineer Words Checker as a simple problem that an advanced, morphologically aware string-matching algorithm – without concern for synonyms – could potentially solve. In summary, Words Checker serves as an in vitro study on LSCG. At the same time, Words Checker has practical applications. Consider a scenario where S is an LLM-generated response y in a conversation, and F consists of words the user explicitly wants to avoid (e.g., when paraphrasing text, for secret keeping, etc.,).

Testing Dataset. To construct a dataset for Words Checker, we use the *CommonGen* (Lin et al., 2020) benchmark, originally designed for traditional constrained text generation. Each entry in CommonGen consists of a sentence and a variable-sized list of W words that are morphologically present in it. For example, an entry may contain "The athlete skied a snowy mountain" with the corresponding words ["ski", "snow"].

We derive our dataset from two partitions of CommonGen, namely the *challenge train sample* and *challenge validation sample*[1]. For these partitions, W ranges from 1 to 4. Given a pool size of candidate forbidden words $|F|$, we: i) construct a vocabulary from all CommonGen partitions, and ii) iterate over the selected partitions to generate valid and invalid samples. To create an *invalid* example, we retain W CommonGen words and randomly sample $|F| - W$ additional vocabulary words. For a *valid* example, we select $|F|$ random words ensuring that none is morphologically present in the sentence.

We generate four versions of Words Checker, each containing 1000 sentences, with increasing constraint complexity: $F = \{10, 100, 500, 1000\}$. We generate balanced datasets, with approximately equal support for both classes. Notice that the 1000 sentences are the same across all scenarios.

4.2 FoCusNet for Words Checker

Model Description. In the practical scenario of Words Checker, we train FoCusNet to recognize whether a sentence S contains a set of words $W = \{w_1, w_2, \ldots, w_n\}$. The training pipeline, summarised in Fig. 3, is divided into three phases:

Phase 1: We use a frozen pre-trained sentence encoder to obtain the initial embeddings for the sentence (e_S) and the words ($\{e_{w_1}, e_{w_2}, \ldots, e_{w_n}\}$).

Phase 2 Next, we refine these embeddings through two learnable projection layers. The sentence embeddings are refined with a linear layer $f_\chi : e_S \rightarrow \hat{e}_S$, where $\hat{e}_S$ is the refined sentence embedding. We aggregate the word embeddings into a single refined embedding $e_{\hat{w}}$ using an attention mechanism (Bahdanau, 2014). Specifically, given the embeddings $e_{w_1}, e_{w_2}, \ldots, e_{w_N}$, we compute $e_{\hat{w}}$ as:

$$e_{\hat{w}} = \sum_{i=1}^{N} f_\gamma(e_{w_i}) \cdot f_\lambda(e_{w_i}) \tag{409}$$

[1]The test partitions of CommonGen do not contain reference sentences.

Intuitively, we use this aggregation layer and focus on more words simultaneously to give the model a broader understanding of the context in which the words are used. For example, with $\{W_1 =$ "mount", "ski"$\}$ and $W_2 =\{$"mount", "lake"$\}$, the model understands that "mount" belongs to both winter- and spring-like scenarios.

We train the layers χ, γ, and λ using the *InfoNCE* loss (Oord et al., 2018), which encourages higher cosine similarities for sentences and words that appear in the same set W. Specifically, two sentences S_1 and S_2 from the same batch are considered positive examples if they share the same set of words, and negative otherwise.

Phase 3: After training the encoder and projection layers, we concatenate the refined sentence embedding $\hat{e}_S$ and the word embedding $e_{\hat{w}}$ into a final embedding $e_f = \hat{e}_S \parallel e_{\hat{w}}$. This concatenated embedding is then fed into a Random Forest classifier, which determines whether the words encoded in $e_{\hat{w}}$ appear in the sentence S or not.

The last two phases of the training pipeline draw inspiration from the *Supervised Contrastive Loss* paper (Khosla et al., 2020), and are designed to learn high-quality embeddings.

Training Dataset. To train FoCusNet, we use the remaining *train* and *validation* partitions from CommonGen. Since more than 80% of the sentences contain a list of three specific words, we apply synthetic augmentation to the dataset. Given a sentence (e.g., "The athlete skied a snowy mountain") with three contained words (e.g.,"athlete", "ski", "mountain"), we randomly select subsets of one (e.g.,"mountain") or two words (e.g.,"athlete", "ski"). The original sentence remains a valid positive sample for each subset. This enhancement allows the model to learn from training examples with varying numbers of words contained, enhancing its generalizability. As we further discuss in Sect.6, note that such augmentations, which exploit logical dependencies, are not specific to this task but generalise across various fields. For example, returning to the example in Fig.2, adopting the appropriate behaviour (e.g., "inviting a Muslim for tea rather than beer") not only aligns with the task "How to be a good visitor" but is also consistent with "How to effectively socialize" and "How to spend quality time with locals while travelling".

Eventually, the final dataset contains $\sim$ 220k labelled examples of sentences and contained words.

5 Experiments

In this section, we present the results of traditional Test Steering Strategies and FoCusNet in Words Checker. While we provide some qualitative insights, our primary focus is on reporting *quantitative metrics* (e.g., accuracy, precision, and recall). A more detailed qualitative analysis, including an examination of specific model responses, can be found in Appendix A.

5.1 Experiments Settings

LLMs Inference. To deploy the LLMs in our Words Checker experiments, we use *SGLang*[2], an open-source framework that facilitates efficient model downloading and deployment. Specifically, we select four models from SGLang's library: *Meta-Llama-3.3-8B-Instruct* and *Meta-Llama-3.3-70B-Instruct* from the LLaMA family (Grattafiori et al., 2024), as well as the more recent *DeepSeek-R1-Distill-Llama-8B* and *DeepSeek-R1-Distill-Llama-70B* from DeepSeek (DeepSeek-AI et al., 2025). The deployment of the 70B models required four NVIDIA RTX A6000 GPUs, whereas the 8B models ran efficiently on a single A6000 GPU. When prompting the models, we set the *temperature* t to 0.2 for the Simple Prompt strategy and increase it to 0.4 for more sophisticated TSS. The exact prompts used are provided in Appendix A.

When using the Best of N strategy, we set N=3.

Training FoCusNet. For the contrastive loss training of FoCusNet, we perform a hyperparameter search using 4-fold cross-validation ($K = 4$), ensuring that all examples sharing the same word list are assigned to the same fold to prevent data leakage. We explore embedding sizes $\{64, 128, 256, 512\}$, learning rates $\{1e^{-4}, 2.5e^{-4}, 5e^{-4}\}$, and InfoNCE loss temperatures $\{0.05, 0.1, 0.2\}$, training for 30 epochs. The best configuration, determined by averaging validation results, consists of an embedding size of 128, a learning rate of $2.5e^{-4}$, a temperature of 0.05, and 24 training epochs, using *all-mpnet-base-v2*[3] as the pre-trained encoder. After selecting the best encoder, we train a random forest where each sentence is paired with a positive (words contained in the sentence) and a negative example (words not contained). A hyperparameter search yields an optimal configuration of 200 trees, a maximum depth

[2] `https://docs.sglang.ai/index.html`

[3] `https://huggingface.co/sentence-transformers/all-mpnet-base-v2`

Table 2: Results of a DeepSeek-R1-Distill-Llama-8B model using different Test Steering Strategies as the number of forbidden words $|F|$ increases. The proposed FoCusNet significantly outperforms other TSS methods.

Test Steering Strategies	$|F|$: 100			$|F|$: 500			$|F|$: 1000		
	Acc.	**Rec.**	**Prec.**	**Acc.**	**Rec.**	**Prec.**	**Acc.**	**Rec.**	**Prec.**
Simple Prompt	86.99	97.25	81.01	70.51	**87.62**	66.33	62.14	82.98	57.52
Chain of Thought	**87.70**	**94.16**	83.88	68.20	87.12	63.03	59.90	78.34	56.83
Best of 3	85.60	94.16	80.94	62.70	83.30	58.81	58.40	80.16	55.46
FoCusNet	87.50	79.18	**95.76**	**79.30**	81.69	**77.78**	**72.80**	**84.01**	**68.26**

of 10, and a minimum of three samples per leaf.

Metrics. Since Words Checker is a standard binary classification problem, we evaluate performance using *accuracy* (overall correctness), *precision* (the proportion of predicted positive sentences that actually contain at least one forbidden word), and *recall* (the proportion of actual positive sentences correctly identified). Additionally, for invalid sentences, we assess the model's parsing ability. To do so, we introduce *parsing precision* and *parsing recall*. For example, given the sentence "The athlete skied the snowy mountain," the set of forbidden words {snow, mountain, ski}, and the model's prediction {snow, ski, sun, fun}, the parsing recall is 0.66 (2 out of 3 correct words retrieved), while the parsing precision is 0.5 (2 out of 4 predicted words are correct).

5.2 Results

Is Words Checker challenging?. We assess the effectiveness of a simple prompting strategy and find that all models, regardless of family or size, experience a roughly 30% accuracy drop as the number of forbidden words increases from 10 to 1000 (see Fig. 4). In addition (full table on Appendix), more forbidden words lead to an increase in false alarms. For example, with 100 forbidden words, LLama 70B has a recall of 97% and precision of 99%, but with 1000 forbidden words, the recall only decreases to 92%, while the precision drops to 65%. These results show that, despite simplicity, Words Checker remains challenging for basic prompting strategies, suggesting that more advanced Test Steering Strategies are needed.

FoCusNet vs. Traditional TSS Limitations. We assess the impact of advanced Test Steering Strategies, like Chain of Thought and Best of 3, on Words Checker using Deepseek's R1-8B model and compare the results with FoCusNet.

Observe the results of Tab. 2. With 100 forbidden words, all methods show similar accuracy. Traditional TSS has better recall, while FoCusNet is more precise. Chain of Thought provides minimal improvement over Simple Prompt, suggesting that the LLM is already following a "Think Step by Step" strategy. The Best of 3 strategy does not help, as, for this simple task, too many opinions lead the final LLM to overthink – even more accentuated in the following scenario. Despite this, the LLM performs adequately in this case, which serves as our reference as we further increase the number of forbidden words.

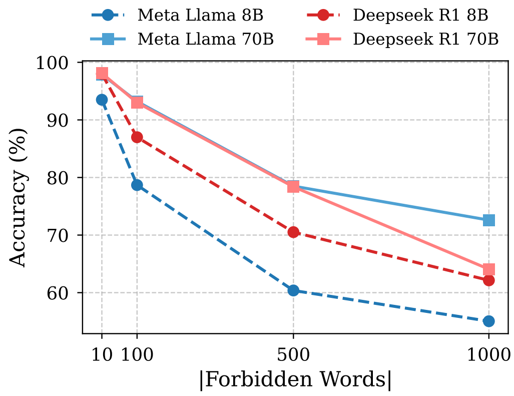

Figure 4: Accuracies with a "Simple Prompt" strategy as the number of forbidden words increases.

With 500 forbidden words, the recall is similar for both traditional Test Steering Strategies and FoCusNet, but FoCusNet achieves +9% higher accuracy due to its better precision. Both Chain of Thought and Best of 3 degrade the performance of Simple Prompt. We find that forcing the model to reason more in simple tasks hinders its performance, as the LLM enters repetitive loops, leading to issues such as: i) confusion between its thought process and the original task, ii) overthinking (e.g.,, "Should I accept synonyms?" or "Do plurals count?"), and iii) hallucination of non-existent words. Contrarily, by focusing on smaller subsets of relevant words (3 for 100 forbidden words, 14 for 500, 30 for 1000), FoCusNet helps the LLM stay on task and reduce false alarms while maintaining a good recall.

Eventually, with 1000 forbidden words the issues observed in the 500-word case are amplified, and

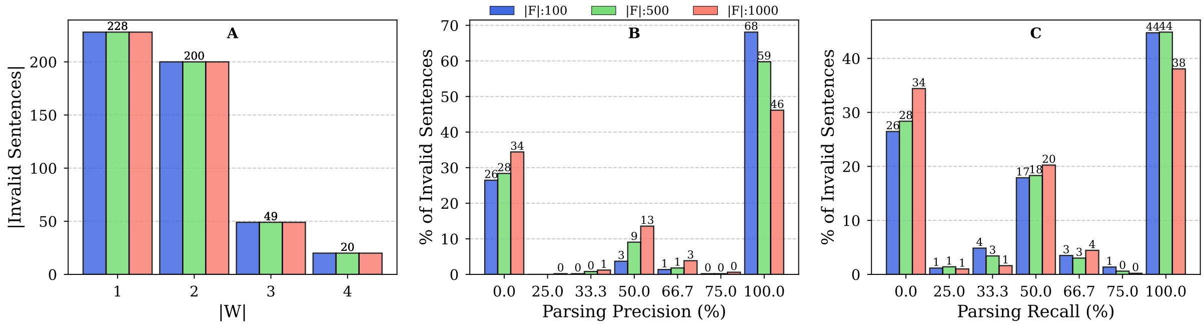

Figure 5: Analysis of recalls and precisions of FoCusNet per invalid sentences

traditional Test Steering Strategies only performs 10% better than random guessing – remember that the problem is balanced. Although FoCusNet performance also declines, it still performs similarly to the 70B-Llama model (68% precision for FoCusNet vs 66% for Llama), which is promising given the ~ 10 times smaller LLM we used here.

Parsing skills of LLM + FoCusNet. Lastly, we conduct a deeper evaluation of our solution, utilizing FoCusNet to enhance the LLM's performance. While the original task was a binary classification – determining whether a sentence was valid or invalid – we now refine our analysis with a more granular approach. Specifically, for invalid sentences, we assess parsing precision by measuring the proportion of predicted words that are actually present in the sentence. Additionally, we evaluate parsing recall by examining how many of the true forbidden words (W) the LLM correctly identifies.

Our analysis focuses on approximately 500 *invalid sentences*, meaning sentences that contain at least one forbidden word ($|W| \geq 1$). This selection allows us to evaluate the detector's ability to identify relevant anomalies.

The results are shown in Fig. 5, with subfigures B and C providing key insights. These subfigures plot the percentage of invalid sentences (*y-axis*) against parsing precision and recall (*x-axis*). For example, they show that when using the list of relevant words identified by FoCusNet, the LLM achieves a parsing precision of 100% for 68% of invalid sentences. Both distributions exhibit a tri-modal pattern, with peaks at 0%, 50%, and 100%. This pattern arises because most invalid sentences in the test dataset contain either one or two forbidden words (as seen in subfigure A).

Although the number of "perfect predictions" (both precise and accurate) consistently exceeds the number of "bogus predictions" (0% precision and recall), increasing the number of candidate words ($|F|$) negatively impacts performance. Notably, the scenarios with $|F| = 100$ and $|F| = 500$ contain the same set of invalid sentences. This means that the true forbidden words (W) in these sentences remain unchanged. For them, FoCusNet always makes the same predictions, irrespective of F. However, as the pool of forbidden candidate words (F) grows, FoCusNet may introduce false positives into the list of relevant words returned to the LLM. These false alarms mislead the LLM, causing it to make more mistakes, thereby reducing overall performance.

6 Conclusions

This paper introduces Large-Scale Constraint Generation (LSCG), a new constrained generation problem where Large Language Models (LLMs) must adhere to a large number of constraints. We designed Words Checker as a controlled testbed of LSCG in which the model classifies sentences as valid or invalid based on an increasingly large list of forbidden words.

Our experiments evaluated models from various families and sizes, testing traditional Test Steering Strategies and introducing FoCusNet, a customizable support module for LLMs. The results highlight a significant performance drop across all models as the number of constraints increases. Standard TSS approaches not only fail to mitigate this decline but often lead models to overthink and hallucinate constraints. In contrast, FoCusNet proves to be the most resilient, consistently improving constraint adherence by narrowing the model's focus.

Despite FoCusNet 's own limitations, its effectiveness in reducing failure rates suggests a promising direction for addressing LSCG. With its simplicity and strong initial results, this study lays the groundwork for future research in constraint-aware LLM reasoning. By defining LSCG and offering open-source implementations of Words Checker and FoCusNet, we aim to inspire the community to explore and benchmark solutions to this critical challenge.

Limitation

In this section, we outline the limitations of the present work.

First, while we provide examples of alternative use cases, we focus solely on a specific instance of Large-Scale Constraint Generation, namely Words Checker. To better isolate the impact of an increasing number of constraints, we deliberately designed Words Checker to minimize the role of the LLM reasoning. Although we believe that this problem has been largely overlooked in prior research, our analysis remains partial, addressing only the complexity of scenarios involving: i) multiple constraints and ii) constraints that require interpretation.

Second, our proposed model, FoCusNet, relies on sufficient task-specific data to perform well. This dependency may limit the applicability of FoCusNet in scenarios where task data are scarce. In the paper, we suggested that augmenting existing datasets through contrastive loss and logical dependencies between constraints and input could mitigate this issue. Additionally, as a task-specific model, FoCusNet does not require extensive generalization, and minor "benign overfitting" is acceptable. Future work should further explore the trade-off between data availability and performance, possibly extending the analysis to contexts beyond Words Checker.

Moreover, while we present FoCusNet as a generic add-on module for LLMs, its architecture has only been evaluated within the Words Checker context. More research is needed to assess its generalizability and explore how different weight architectures might affect its performance.

Finally, our work has concentrated solely on textual constraints. However, in many real-world tasks, constraints may span multiple modalities (Chi et al., 2024; Inan et al., 2023). Future research could address the challenges posed by the large number of constraints in different modalities. In this regard, FoCusNet could offer valuable flexibility, as it could be adapted with modality-specific architectures to better address these challenges.

References

Muhammad F. Adilazuarda, Sagnik Mukherjee, and Pradhyumna et al. Lavania. 2024. Towards measuring and modeling "culture" in llms: A survey. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Miami, Florida, USA. Association for Computational Linguistics.

Dzmitry Bahdanau. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Yushi Bai, Xin Lv, and Jiajie et al. Zhang. 2024. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, Bangkok, Thailand. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, and Nick et al. Ryder. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*.

Shuo Chang, F. Maxwell Harper, and Loren G. Terveen. 2016. Crowd-based personalized natural language explanations for recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA. Association for Computing Machinery.

Lichang Chen, Shiyang Li, and Jun Yan et al. 2024a. Alpagasus: Training a better alpaca with fewer data. In *The Twelfth International Conference on Learning Representations*.

Xinyun Chen, Maxwell Lin, and Nathanael et al. Sch"arli. 2024b. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*.

Jianfeng Chi, Ujjwal Karn, and Hongyuan et al. Zhan. 2024. Llama Guard 3 Vision: Safeguarding Human-AI Image Understanding Conversations. *arXiv preprint arXiv:2411.10414*.

Hyung Won Chung, Le Hou, and Shayne et al. Longpre. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

Karl Cobbe, Vineet Kosaraju, and Mohammad et al. Bavarian. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

DeepSeek-AI, Daya Guo, and Dejian et al. Yang. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Gelei Deng, Yi Liu, and V'ictor et al. Mayoral-Vilches. 2024. Pentestgpt: Evaluating and harnessing large language models for automated penetration testing. In *33rd USENIX Security Symposium*, Philadelphia, PA. USENIX Association.

Haikang Deng and Colin Raffel. 2023. Reward-augmented decoding: Efficient controlled text generation with a unidirectional reward model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 11781–11791, Singapore. Association for Computational Linguistics.

Aaron Grattafiori, Abhimanyu Dubey, and Abhinav et al. Jauhri. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Ahmed E. Hassan, Dayi Lin, and Gopi K. et al. Rajbahadur. 2024. Rethinking software engineering in the era of foundation models: A curated catalogue of challenges in the development of trustworthy fmware. In *Companion Proceedings of the 32nd ACM International Conference on Software Engineering*, New York, NY, USA. Association for Computing Machinery.

Qianyu He, Jie Zeng, Qianxi He, Jiaqing Liang, and Yanghua Xiao. 2024a. From complex to simple: Enhancing multi-constraint complex instruction following ability of large language models. *arXiv preprint arXiv:2404.15846*.

Qianyu He, Jie Zeng, and Qianxi et al. He. 2024b. From complex to simple: Enhancing multi-constraint complex instruction following ability of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10864–10882.

Qianyu He, Jie Zeng, and Wenhao et al. Huang. 2024c. Can large language models understand real-world complex instructions? In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence*. AAAI Press.

Zhiting Hu, Zichao Yang, and Xiaodan et al. Liang. 2017. Toward controlled generation of text. In *Proceedings of the 34th International Conference on Machine Learning*. PMLR.

Zhiting Hu, Zichao Yang, and Xiaodan et al. Liang. 2018. Toward controlled generation of text. *arXiv preprint arXiv:1703.00955*.

Hakan Inan, Kartikeya Upasani, and Jianfeng et al. Chi. 2023. Llama Guard: LLM-based Input-Output Safeguard for Human-AI Conversations. *arXiv preprint arXiv:2312.06674*.

Prannay Khosla, Piotr Teterwak, and Chen et al. Wang. 2020. Supervised contrastive learning. *Advances in neural information processing systems*, 33.

Taeyoun Kim, Suhas Kotha, and Aditi Raghunathan. 2024. Testing the Limits of Jailbreaking Defenses with the Purple Problem. *arXiv preprint*. ArXiv:2403.14725 [cs].

Hadas Kotek, Rikker Dockum, and David Sun. 2023. Gender bias and stereotypes in large language models. In *Proceedings of The ACM Collective Intelligence Conference*, New York, NY, USA. Association for Computing Machinery.

Ben Krause, Akhilesh Deepak Gotmare, Bryan McCann, Nitish Shirish Keskar, Shafiq Joty, Richard Socher, and Nazneen Fatema Rajani. 2021. GeDi: Generative Discriminator Guided Sequence Generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4929–4952, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Mo Li, Songyang Zhang, and Yunxin et al. Liu. 2024. Needlebench: Can llms do retrieval and reasoning in 1 million context window? *arXiv preprint arXiv:2407.11963*.

Hunter Lightman, Vineet Kosaraju, and Yuri et al. Burda. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Bill Yuchen Lin, Ming Shen, and Wangchunshu et al. Zhou. 2020. CommonGen: A constrained text generation challenge for generative commonsense reasoning. In *Automated Knowledge Base Construction*.

Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A. Smith, and Yejin Choi. 2021. DExperts: Decoding-time controlled text generation with experts and anti-experts. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*, Online. Association for Computational Linguistics.

Michael X. Liu, Frederick Liu, and Alexander J. et al. Fiannaca. 2024a. "we need structured output": Towards user-centered constraints on large language model output. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, New York, NY, USA. Association for Computing Machinery.

Tianqi Liu, Yao Zhao, and Rishabh Joshi et al. 2024b. Statistical rejection sampling improves preference optimization. In *The Twelfth International Conference on Learning Representations*.

Aman Madaan, Niket Tandon, and Prakhar et al. Gupta. 2023. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Swaroop Mishra, Daniel Khashabi, and Chitta et al. Baral. 2022. Cross-task generalization via natural language crowdsourcing instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*.

Norman Mu, Sarah Chen, and Zifan et al. Wang. 2024. Can LLMs follow simple rules? *arXiv preprint arXiv:2311.04235*.

Reiichiro Nakano, Jacob Hilton, and Suchir et al. Balaji. 2022. WebGPT: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Tarek Naous, Michael J. Ryan, and Wei et al. Xu. 2024. Having beer after prayer? measuring cultural bias in large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, Bangkok, Thailand. Association for Computational Linguistics.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

Long Ouyang, Jeff Wu, and Xu et al. Jiang. 2022. Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA. Curran Associates Inc.

Traian Rebedea, Razvan Dinu, and Makesh et al. Sreedhar. 2023. NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications with Programmable Rails. *arXiv preprint arXiv:2310.10501*.

Mrinank Sharma, Meg Tong, Jesse Mu, Jerry Wei, and et al. 2025. Constitutional classifiers: Defending against universal jailbreaks across thousands of hours of red teaming. *arXiv preprint arXiv:2501.18837*. ArXiv:2501.18837 [cs].

Weijia Shi, Sewon Min, and Michihiro et al. Yasunaga. 2024. REPLUG: Retrieval-augmented black-box language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics*.

Robert J. Sternberg. 1986. Beyond iq: A triarchic theory of human intelligence. *British Journal of Educational Studies*, 34(2):205–207.

Nisan Stiennon, Long Ouyang, and Jeffrey et al. Wu. 2020. Learning to summarize with human feedback. In *Advances in Neural Information Processing Systems*.

Jiao Sun, Yufei Tian, and Wangchunshu et al. Zhou. 2023. Evaluating large language models on controlled generation tasks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Singapore. Association for Computational Linguistics.

Romal Thoppilan, Daniel De Freitas, and Jamie et al. Hall. 2022. LaMDA: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*.

Changjie Wang, Mariano Scazzariello, and Alireza et al. Farshin. 2024. Netconfeval: Can llms facilitate network configuration? *Proceedings of the ACM on Networking*, 2(CoNEXT2).

Yizhong Wang, Hamish Ivison, and Pradeep et al. Dasigi. 2023. How far can camels go? exploring the state of instruction tuning on open resources. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Zhaoyang Wang, Jinqi Jiang, and Huichi et al. Zhou. 2025. Verifiable format control for large language model generations. *arXiv preprint arXiv:2502.04498*.

Jason Wei, Yi Tay, and Rishi et al. Bommasani. 2022a. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *arXiv preprint*. ArXiv:2201.11903 [cs].

Jason Wei, Xuezhi Wang, and Dale et al. Schuurmans. 2022b. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*.

Congying Xia, Chen Xing, and Jiangshu et al. Du. 2024. Fofo: A benchmark to evaluate llms' format-following capability. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, Bangkok, Thailand. Association for Computational Linguistics.

Kevin Yang and Dan Klein. 2021. FUDGE: Controlled text generation with future discriminators. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics*.

Shunyu Yao, Howard Chen, and Austin W. et al. Hanjie. 2024. Collie: Systematic construction of constrained text generation tasks. In *The Twelfth International Conference on Learning Representations*.

Qi Zhang, Yiming Zhang, Haobo Wang, and Junbo Zhao. 2024. RECOST: External knowledge guided data-efficient instruction tuning. In *Findings of the Association for Computational Linguistics: ACL 2024*, Bangkok, Thailand. Association for Computational Linguistics.

Yongfeng Zhang and Xu Chen. 2020. Explainable recommendation: A survey and new perspectives. *Foundations and Trends in Information Retrieval*, 14(1):1–101.

Hao Zhao, Maksym Andriushchenko, Francesco Croce, and Nicolas Flammarion. 2024. Long is more for alignment: a simple but tough-to-beat baseline for instruction fine-tuning. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Chunting Zhou, Pengfei Liu, and Puxin Xu et al. 2023. LIMA: Less is more for alignment. In *Thirty-seventh Conference on Neural Information Processing Systems*.

A Appendix

A.1 LLM prompts

We here provide the prompts we used for the LLM inference:

Simple Prompt.“Check if the following sentence contains one of the following set of words. Only answer True or False. Ensure to include your final answer into <answer></answer>. For instance, if the sentence contains one of the words, answer <answer>True</answer>; <answer>False</answer> otherwise.”

Chain of Thought.“Your task is to check if the sentence contains one of the following set of words. Answer True or False. Please, make sure to think step by step. Ensure to include your final answer into <answer></answer>. For instance, if the sentence contains one of the words, answer <answer>True</answer>; <answer>False</answer> otherwise.”

Best of 3.:

– **For the judges**, we use: “Check if the following sentence contains one of the following set of words. Do not include your reasoning process in the answer; Provide a short explanation (at most 100 words) to justify your answer. Conclude your sentence with <answer>your answer</answer>, where your answer is either True or False.”

– **For the final verdict**: “Give me your final opinion over the verdicts of a jury of {args.n_rounds} LLMs. When prompted the following message: <message> A jury of LLMs answered: <answer> What is your final verdict? Just answer True or False: Ensure to include your final answer into <answer></answer>. For instance, if the sentence contains one of the words, answer <answer>True</answer>; <answer>False</answer> otherwise.”

A.2 Examples of LLMs answers

We here report some of the erroneous answers the LLM might return.

Hallucinations. Here, the model thinks that the word “dog” is present in the list of forbidden words when processing the sentence “dressed my dog up as a pirate”. However, the word is not present – even if the conviction of the model would suggest so.

Ovethinking #1. Here, the LLM, prompted to “Think Step by Step”, initially gets the correct answer – i.e., none of the words is present in the sentence. However, it is unsure about the answer, and in a second pass it eventually self-convince that all words are present.

Overthinking #2. This is an example from the Best of 3 strategy. Two support models agree that the sentence does not contain any words. However, the third model advocates for the devil, saying that “adorn” is actually a synonym for “dress”, and that, therefore, the sentence shall be labelled as invalid. The final judge model trust this third model – as it can provide a convincing explanation – and (erroneously) triggers the alert.

Hallucination

Okay, so I need to figure out which words from the given list are in the sentence: "dressed my dog up as a pirate." The task is to report all the words that are contained in the sentence. If none are found, I should just list an empty list.

First, I'll read the sentence carefully: "dressed my dog up as a pirate." I need to identify all the individual words in this sentence and see which ones match any from the provided list.

Let me break down the sentence word by word. The first word is "dressed." I should check if "dressed" is in the list of words. Looking through the list, I don't see "dressed" there. So that's one word not present.

Next is "my." Checking the list, "my" isn't there either. Moving on.

The third word is "dog." I'll look for "dog" in the list. Hmm, I don't see it either. So "dog" isn't present.

Then we have "up." Scanning the list, "up" isn't listed. So that's another word not found.

The next word is "as." Checking the list, "as" isn't there. So that's not present.

Finally, "a pirate." So "a" and "pirate." Checking the list, "a" isn't there, but "pirate" is in the list. Wait, let me double-check. The list includes "pirate" as one of the words. So "pirate" is present.

Wait, but the sentence says "as a pirate," so "a" is part of that phrase, but "a" isn't in the list. So only "pirate" is in the list.

Let me go through the list again to make sure I didn't miss any. The words provided are quite extensive, but I only found "pirate" in the sentence.

So, the only word from the list that's in the sentence is "pirate."
</think>

<answer> True </answer>

Overthinking

Okay, so I've got this task where I need to check which words from a given list are present in a specific sentence. The sentence is: "winners podium with numerals on the sky". The words provided are quite a long list, so I need to make sure I go through each one carefully.

First, I'll read the sentence again to understand the context. It mentions "winners", "podium", "numerals", and "sky". I need to see which of the given words match any of these terms or are related in meaning.

Looking at the list of words, I'll go through them one by one. Starting with "headlamp" – doesn't seem relevant. "Success" – maybe, but the sentence doesn't explicitly mention success. "Wrestle" – no, that's unrelated. "Crease" – not here. "Poultry" – definitely not. "Softball" – nope. "Rodeo" – no. "Bait" – not present. "Therapy" – doesn't fit. "Whistle" – no. "Navigate" – not in the sentence. "Aim" – maybe, but not directly. "Attach" – no. "Melody" – no. "Mock" – no. "Pasture" – no. "Pass" – possibly, but not directly. "College" – no. "Header" – no. "Walker" – no. "Resident" – no. "Calm" – maybe, but not in context. "Tiger" – no. "Hunter" – no. "Ambush" – no. "Motorboat" – no. "Position" – maybe, but not clearly. "Squeeze" – no. "Lease" – no. "Bubble" – no. "Wrap" – no. "Bounce" – no. "Teller" – no. "Pattern" – no. "Headquarters" – no. "Brightness" – no. "Ballroom" – no. "Birth" – no. "Statue" – no. "Submarine" – no. "Kiss" – no. "Indoor" – no. "Entry" – no. "Exhibition" – no. "Proceed" – maybe, but not directly. "Prey" – no. "Puffin" – no. "Friend" – no. "Swoop" – no. "Asteroid" – no. "Flag" – no. "Hamburger" – no. "Stool" – no. "Beg" –

```
    no. "Resource" – no. "Electricity" – no. "Backseat" – no. "Anxiety" – no.
     "Shadow" – no. "Lego" – no. "Uniform" – no. "Earn" – no. "Shepherd" – no
    . "Marathon" – no. "Liquid" – no. "Theme" – no. "Bomber" – no. "
    Thanksgive" – no. "Request" – no. "Resistance" – no. "Climb" – no. "
    Cantilever" – no. "Bag" – no. "Push" – no. "Projector" – no. "Undergo" –
    no. "Accompany" – no. "Characteristic" – no. "Nursery" – no. "Salt" – no.
     "Mull" – no. "Workman" – no. "Sunset" – no. "Engrave" – no. "Slice" – no
    . "Crop" – no. "Clarinet" – no. "Kill" – no. "Blouse" – no. "Purple" – no
    . "Disappoint" – no. "Segment" – no. "Shoal" – no. "Memo" – no. "Cite" –
    no. "Hope" – no. "Deforestation" – no. "Dial" – no. "Tree" – no. "
    Militant" – no.

Wait, I think I might have missed "podium". Let me check again. Yes, "podium"
     is in the sentence. So that's one word. Also, "winners" is another. "
    Numerals" is there too. "Sky" is mentioned as well. Let me make sure none
     of the other words are in the sentence. "With" is a preposition, but it'
    s not listed as a word to check. So the words in the sentence that match
    the list are winners, podium, numerals, and sky.

Wait, but the original list includes "sky" as one of the words. So I should
    include that. So the words are winners, podium, numerals, and sky.
</think>
<answer> True </answer>
```

Overthinking #2

Give me your final opinion over the verdicts of a jury of 3 LLMs.

When prompted the following message: Task: Check if the following sentence contains one of the following set of words. Do not include your reasoning process in the anser; Provide a short explanation (at most 100 words) to justify your answer. Conclude your sentence with <answer>your answer</answer>, where your answer is either True or False.

Sentence: dressed my dog up as a pirate .

Words: [massage, container, length, hairstyle, mountain, lesson, shop, boutique, glider, iron, steal, orangutan, overall, blur, lava, equestrian, exert, hear, scuba, slice, manufacturing, personality, decor, cemetery, plantation, ear, excite, argue, inscription, cook, deity, pallet, skyline, hose, chromosome, awning, contrail, brand, right, medina, catwalk, sheer, berth, lamb, complain, impress, ignore, onion, thrive, smell, guitarist, effect, header, aim, ask, dome, panini, headdress, table, adorn, evacuate, sparkle, girlfriend, synchrony, export, tag, puff, school, fiddle, queue, beam, medal, intensify, distance, alm, bullfight, midair, mangrove, bless, wall, mow, universe, candle, galaxy, reel, soar, worship, sneeze, gallop, clank, chimney, helmet, allow, leather, embrace, iceberg, lantern, stride, vacation, wallow]

A jury of LLMs answered:

Judge 0: <think>

Okay, so I need to check if the sentence "dressed my dog up as a pirate" contains any of the given words. Let me go through each word one by one to make sure I don't miss any.

First, the sentence is about dressing a dog as a pirate. So, the main elements are dressing, dog, pirate. Now, looking at the list of words, I see words like "adorn," which means to decorate or dress up. That's a possibility because the sentence is about dressing the dog.

Next, I'll check the other words to be thorough. "Container" doesn't fit here. "Length" and "hairstyle" are unrelated. "Mountain," "lesson," "shop," "boutique," "glider," "iron," "steal," "orangutan," "overall," "blur," "lava," "equestrian," "exert," "hear," "scuba," "slice," "manufacturing," "personality," "decor," "cemetery," "plantation," "ear," "excite," "argue," "inscription," "cook," "deity," "pallet," "skyline," "hose," "chromosome," "awning," "contrail," "brand," "right," "medina," "catwalk," "sheer," "berth," "lamb," "complain," "impress," "ignore," "onion," "thrive," "smell," "guitarist," "effect," "header," "aim," "ask," "dome," "panini," "headdress," "table," "adorn," "evacuate," "sparkle," "girlfriend," "synchrony," "export," "tag," "puff," "school," "fiddle," "queue," "beam," "medal," "intensify," "distance," "alm," "bullfight," "midair," "mangrove," "bless," "wall," "mow," "universe," "candle," "galaxy," "reel," "soar," "worship," "sneeze," "gallop," "clank," "chimney," "helmet," "allow," "leather," "embrace," "iceberg," "lantern," "stride," "vacation," "wallow."

Wait, I already found "adorn" as a possible match. Let me make sure there's no other word that might fit. "Adorn" is the only one that relates to dressing or decorating something. The sentence is about dressing the dog, so "adorn" fits perfectly. The other words don't seem to connect at all. So, the sentence does contain one of the specified words.

</think>

The sentence contains the word "adorn," which means to decorate or dress something. Since the sentence is about dressing a dog as a pirate, "adorn" fits the context. <answer>True</answer>

Judge 1: <think>

Alright, so I need to check if the sentence "dressed my dog up as a pirate" contains any of the given words. Let me go through each word one by one. First, I'll read the sentence carefully. It says the user dressed their dog up like a pirate. So, the main actions here are dressing and being a pirate.

Looking at the list of words, I see words like 'dress' or 'clothing' aren't

```
    present. The sentence mentions 'dog' and 'pirate', but those aren't in
    the list either. I'll go through each word from the list to make sure I
    don't miss anything. Words like 'massage', 'container', 'length', '
    hairstyle' - none of these are in the sentence. 'Mountain', 'lesson', '
    shop', 'boutique' - nope. 'Glider', 'iron', 'steal', 'orangutan' - still
    nothing. 'Overall', 'blur', 'lava', 'equestrian', 'exert', 'hear', 'scuba
    ', 'slice', 'manufacturing', 'personality', 'decor', 'cemetery', '
    plantation', 'ear', 'excite', 'argue', 'inscription', 'cook', 'deity', '
    pallet', 'skyline', 'hose', 'chromosome', 'awning', 'contrail', 'brand',
    'right', 'medina', 'catwalk', 'sheer', 'berth', 'lamb', 'complain', '
    impress', 'ignore', 'onion', 'thrive', 'smell', 'guitarist', 'effect', '
    header', 'aim', 'ask', 'dome', 'panini', 'headdress', 'table', 'adorn', '
    evacuate', 'sparkle', 'girlfriend', 'synchrony', 'export', 'tag', 'puff',
     'school', 'fiddle', 'queue', 'beam', 'medal', 'intensify', 'distance', '
    alm', 'bullfight', 'midair', 'mangrove', 'bless', 'wall', 'mow', '
    universe', 'candle', 'galaxy', 'reel', 'soar', 'worship', 'sneeze', '
    gallop', 'clank', 'chimney', 'helmet', 'allow', 'leather', 'embrace', '
    iceberg', 'lantern', 'stride', 'vacation', 'wallow'.

After going through each word, I don't see any that match the sentence. The
    sentence uses 'dressed' and 'dog', but those aren't in the list. The
    closest might be 'adorn', but that's not present either. So, the sentence
     doesn't contain any of the specified words.
</think>

<answer>False</answer>
Judge 2: <think>
Okay, so I need to check if the sentence "dressed my dog up as a pirate"
    contains any of the given words. Let me list out the words first to make
    sure I don't miss any. The words are: massage, container, length,
    hairstyle, mountain, lesson, shop, boutique, glider, iron, steal,
    orangutan, overall, blur, lava, equestrian, exert, hear, scuba, slice,
    manufacturing, personality, decor, cemetery, plantation, ear, excite,
    argue, inscription, cook, deity, pallet, skyline, hose, chromosome,
    awning, contrail, brand, right, medina, catwalk, sheer, berth, lamb,
    complain, impress, ignore, onion, thrive, smell, guitarist, effect,
    header, aim, ask, dome, panini, headdress, table, adorn, evacuate,
    sparkle, girlfriend, synchrony, export, tag, puff, school, fiddle, queue,
     beam, medal, intensify, distance, alm, bullfight, midair, mangrove,
    bless, wall, mow, universe, candle, galaxy, reel, soar, worship, sneeze,
    gallop, clank, chimney, helmet, allow, leather, embrace, iceberg, lantern
    , stride, vacation, wallow.

Hmm, the sentence is about dressing a dog as a pirate. So, the main elements
    are "dog," "dress," "pirate." Now, looking through the list, I don't see
    "dog" listed, so that's a point against it. But wait, the words are about
     checking if any of them are present in the sentence, not necessarily the
     other way around.

Wait, the sentence is "dressed my dog up as a pirate." So the words used are
    "dressed," "my," "dog," "up," "as," "a," "pirate." None of those words
    are in the provided list. Let me double-check the list again. The words
    are all nouns, I think. So "dressed" is a verb, "dog" is a noun, but "dog
    " isn't in the list. So the sentence doesn't contain any of the specified
     words. Therefore, the answer should be False.
</think>

<answer>False</answer>
What is your final verdict? Just answer True or False: Ensure to enclude your
     final answer into <answer></answer>. For instance, if the sentence
    contains one of the words, answer <answer>True</answer>; <answer>False</
    answer> otherwise.
(Pdb) answer_critic
'<think>\nAlright, so I need to determine if the sentence "dressed my dog up
    as a pirate" contains any of the specified words. Let me go through each
    word in the list to see if any match.\n\nFirst, the sentence is about
    dressing a dog as a pirate. The key elements are "dress," "dog," and "
    pirate." Now, looking at the list of words provided: massage, container,
    length, hairstyle, mountain, lesson, shop, boutique, glider, iron, steal,
     orangutan, overall, blur, lava, equestrian, exert, hear, scuba, slice,
```

```
manufacturing, personality, decor, cemetery, plantation, ear, excite,
argue, inscription, cook, deity, pallet, skyline, hose, chromosome,
awning, contrail, brand, right, medina, catwalk, sheer, berth, lamb,
complain, impress, ignore, onion, thrive, smell, guitarist, effect,
header, aim, ask, dome, panini, headdress, table, adorn, evacuate,
sparkle, girlfriend, synchrony, export, tag, puff, school, fiddle, queue,
 beam, medal, intensify, distance, alm, bullfight, midair, mangrove,
bless, wall, mow, universe, candle, galaxy, reel, soar, worship, sneeze,
gallop, clank, chimney, helmet, allow, leather, embrace, iceberg, lantern
, stride, vacation, wallow.\n\nAfter reviewing each word, I notice that "
adorn" is present in the list. The word "adorn" means to decorate or
dress something. Since the sentence is about dressing a dog, "adorn" fits
 the context. Therefore, the sentence does contain one of the specified
words.\n<answer>True</answer>'
```