Combining Constraint Programming Reasoning with Large Language Model Predictions

Florian Régin
Université Côte d'Azur, I3S, CNRS, Sophia Antipolis, France

Elisabetta De Maria
Université Côte d'Azur, I3S, CNRS, Sophia Antipolis, France

Alexandre Bonlarron
Université Côte d'Azur, Inria, Sophia Antipolis, France
Université Côte d'Azur, I3S, CNRS, Sophia Antipolis, France

Abstract

Constraint Programming (CP) and Machine Learning (ML) face challenges in text generation due to CP's struggle with implementing "meaning" and ML's difficulty with structural constraints. This paper proposes a solution by combining both approaches and embedding a Large Language Model (LLM) in CP. The LLM handles word generation and meaning, while CP manages structural constraints. This approach builds on GenCP, an improved version of On-the-fly Constraint Programming Search (OTFS) using LLM-generated domains. Compared to Beam Search (BS), a standard NLP method, this combined approach (GenCP with LLM) is faster and produces better results, ensuring all constraints are satisfied. This fusion of CP and ML presents new possibilities for enhancing text generation under constraints.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases Solver and Tools, ML-augmented CP, Constrained Text Generation, ML alongside CO

Digital Object Identifier 10.4230/LIPIcs.CP.2024.25

Acknowledgements We thank Jack Massey for his help in reproducing the benchmarks used as baseline in Section 4.1.1.

1 Introduction

How can we perceive Constraint Programming beyond its traditional role in solving combinatorial optimization problems? Once Eugene Freuder wrote *Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it* [13].

Nevertheless, some real-world problems are still beyond the reach of the current CP paradigm. This is particularly true when real-world problems involve vague notions such as "meaning" and "melody" for text and music. These are not easy to model in CP with the classical toolbox, mainly because these notions are hard to define formally. For instance, it is unclear how to formalize an objective function or a constraint to get closer to a meaningful sentence, a melodious song or a captivating painting. On the other hand, recent results in Machine Learning (ML), such as transformer-based models [39], have demonstrated the power of these techniques to capture a significant part of these vague concepts through data-driven statistical learning (e.g., Large Language Model (LLM) like the GPT series [8], stable-diffusion [33], ChatMusician [41]). In the article, we demonstrate that ML, and in particular LLM, can help CP to model and solve problems where such vague concepts can be found.

30th International Conference on Principles and Practice of Constraint Programming (CP 2024).
Editor: Paul Shaw; Article No. 25; pp. 25:1–25:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In recent years, there has been a growing interest in text generation under constraints thanks to the rise of transformer-based models, like OpenAI ChatGPT ([8]) and Meta LLaMa ([37]). Nevertheless, even fine-tuned prompted LLMs fail to generate several constrained outputs (see the tasks introduced in [40]). The goal of this paper is to present a new method for the task of text generation under constraints. This interest has a strong chance of continuing to grow insofar as many brands wish to integrate these technologies, in particular with their customers, and want to have control and guarantees on the behavior of these conversational agents. Hence, it may impact several critical marketing aspects (e.g., brand representation, legal issues, data privacy, . . .). Therefore, CP has the potential to become a strong safeguard of this kind of generative model.

For the task of text generation under constraints, ML techniques face limitations when they have to manage structural constraints, such as limits on the number of words or characters (e.g. Text Summarization, Text Simplification, Text style transfer, Question Answering, Storytelling, Poetry or Lyrics Generation, Subtitle) [15]. CP succeeds on these types of constraints, making the combination of CP and ML a natural fit for the task of text generation under constraints.

This paper proposes such a combination, to tackle a class of problems where neither CP and ML succeeds on their own (Fig. 1).

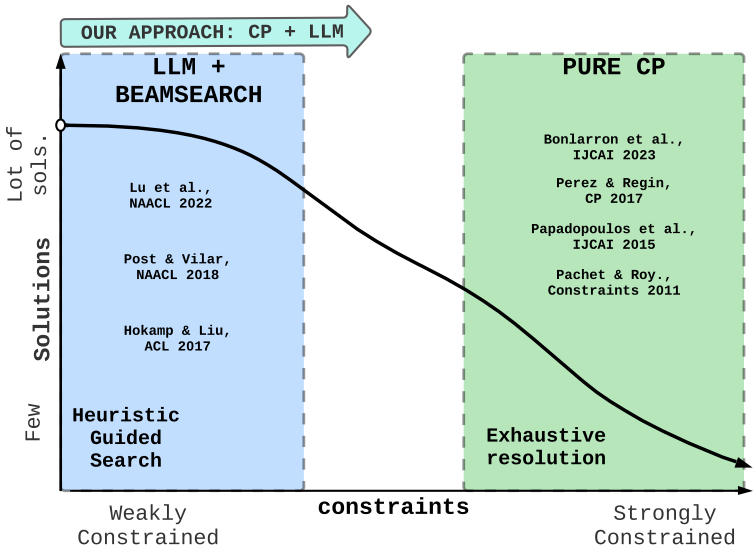

Figure 1 Our approach aspires to explore the in-between area. In the blue (left-hand side) region, LLM guided searches solve weakly constrained problems [27, 32, 17] and in the green (right-hand side) region, CP-based generation tackles strongly constrained problems [36, 5, 6, 4, 31, 30, 29].

Combining Combinatorial Optimization (CO) and ML is a very active research area [2], however there is no easy way to integrate the ML "expertise" into CP as a constraint of the model [1, 26] and *vice versa* [18]. Furthermore, there are many incentives to strengthen the interactions between ML and CO [21, 22, 35]. Usually, the main motivation comes from the performance perspective, where the idea is to improve a solver's performance with ML (e.g., finding branching heuristics thanks to Reinforcement Learning [9] or finding better bounds with Clustering [28]). This paper tackles it from the modeling point of view. Modeling is a crucial part of CO works. In the end, the model must account for the underlying solver that runs it. More in detail, here, the paper focuses on the interaction between CP and ML, more precisely through an ML-augmented CP approach [23].

In the context of text generation under constraints, the domain of a variable represents a word. The base idea of the paper consists in letting ML manage the domain of variables and CP manage the constraints and the number of variables. In this manner, the sentence formed by variables has high chances to have a meaning and all the constraints will be satisfied. In traditional CP, the domains can not be managed by ML because they have to be set beforehand. However, it is possible to rely on On-the-fly Constraint Programming Search (OTFS) [34], a CP based method where variables, domains and constraints are generated during the search for a solution.

The main contribution of this paper is to propose a new version of OTFS, called GenCP, where the generative function of the domain of variables is modified to allow CP variable domains to be computed by an LLM embedded in it, during the search for a solution. More in detail, ML is used during process solving but it is also used as an explicit part of the problem definition (i.e., domains are predicted by the LLM and can replace entirely static variable domains definition of a CSP.). Thus it bridges CP and ML through solving and modelling.

The potential of the approach is showcased for the problem of text generation under constraints, against one the most used techniques in the Natural Language Processing (NLP) field: Beam Search (BS). Both methods (BS and GenCP) are compared on constrained sentences generation tasks extracted from benchmarks recently introduced [40]. The approach highlights how CP can provide guarantees when combined with LLM predictions.

The paper is organized as follows: Sec. 2 serves as background, Sec. 3 shows how to extend OTFS to GenCP and how to implement an interaction between GenCP and LLM. Sec. 4 presents the experimental results in which the new approach is demonstrated on the task of text generation under constraints. Finally, Sec. 5 delves into further discussion, offering additional insights into this work and providing perspectives for future research endeavors.

2 Background

This section introduces the necessary background on LLM and CP.

2.1 LLM Predictions Strategies

2.1.1 Decoding Strategies Combined with LLMs

Large Language Models (LLMs), such as the GPT series, generate text by predicting the next token (word or character) given the history of previously generated words. Decoding in LLMs refers to the strategy used to select the next words to be generated.

2.1.2 Greedy Decoding

The simplest decoding strategy is greedy decoding. Here, the LLM selects the words with the highest probability at each time step. Although simple and efficient, this approach does not guarantee the best overall sequence, as it does not consider the effect of the current selection on future tokens.

2.1.3 Beam Search

Beam Search (BS) [25, 32, 17] is a refined version of greedy decoding. A beam is a candidate sequence of words. Instead of selecting the single best token at each time step, it usually keeps track of the k most likely sequences (beams) at each step.

Although BS usually achieves better results than greedy decoding, it assumes that high-ranking token sequences consist of high-ranking tokens, which may only sometimes be the case. For a more stochastic and diverse output, top-k sampling and top-p sampling (also known as nucleus sampling) are used. In top-k sampling, the model selects from the top k highest probability predictions, while in top-p sampling, it dynamically selects the number of top predictions to cover p percent of the total probability mass.

2.1.4 Perplexity

Perplexity is an entropy metric derived from Shannon's information theory [7]. Since an LLM computes the probability of text, then it can compute text perplexity. It can be expressed as the geometric mean of the inverse conditional likelihood of the sequence [20]. Let S_n be the sequence of a succession of words of size n: $S_n = w_1 w_2 .. w_n$. The perplexity (PPL) of S_n is computed as follows:

$$PPL(S_n) = \sqrt[n]{\frac{1}{P(w_1 w_2 w_3 ... w_n)}},$$

where probability $P(\cdot)$ is given by the LLM. PPL can be interpreted as the "how likely a text is generated by a given model" [15]. Usually, it is used to evaluate the LLM itself by checking that good samples are recognized as such (i.e., low PPL values).

In NLP, the evaluation of text is still an open problem, and human evaluation remains the gold standard. Numerous metrics have been developed to address this issue. Among them, PPL remains an objective criterion associated with text produced by a given model. PPL is also much more convenient to use than pure probability. Its range is $[1; +\infty[$. The lower, the better.

2.2 Constraint Programming

Constraint Programming (CP) is a paradigm for solving combinatorial problems that draws on a wide range of techniques from artificial intelligence and operations research. In CP a problem can be defined as a Constraint Satisfaction Problem (CSP). A CSP is a triplet: $\langle X, D, C \rangle$, where:

- $X = \{X_1, X_2, ..., X_n\}$ is the set of variables of the problem.
- $D = \{D_{X_1}, D_{X_2}, ..., D_{X_n}\}$ is the set of domains, where each domain D_{X_i} corresponds to the set of possible values for the variable X_i.
- $C = \{c_1, c_2, ..., c_m\}$ is the set of constraints of the problem. A constraints represent a property of the problem.

A solution is an assignment of all the variables to a value present in their respective domains, such that all the constraints are satisfied.

2.2.1 Avoiding Static Definition of the CSP

In traditional CP, for the task of text generation under constraints, a variable represents a word. Since the domains of variables have to be set beforehand, they will be of enormous size, containing every word/declination of words for a given language. Furthermore, constraints between succession of words may lead to a combinatorial explosion. Since traditional CP is not well suited, this work focuses on OTFS, a CP based method recently introduced by Régin and De Maria [34]. Instead of having the variables/domains/constraints set before the search, OTFS generates the variables/domains/constraints during the search for a solution, avoiding the problem stated above and being expendable to permit the integration of an LLM. The new version of OTFS is called GenCP.

3 Method: LLM alongside OTFS

The approach of this paper extends OTFS by having an embedded LLM generate the domains of variables. Figure 2 graphically depicts the interplay between those components. The approach also adds a minor improvement in the form of helping functions, to differentiate between implicit constraints (prevent infinite loops, ensure a variable represents a word, etc.) and explicit constraints (constraints of the problem). In the next subsection, the new version of OTFS called GenCP is described.

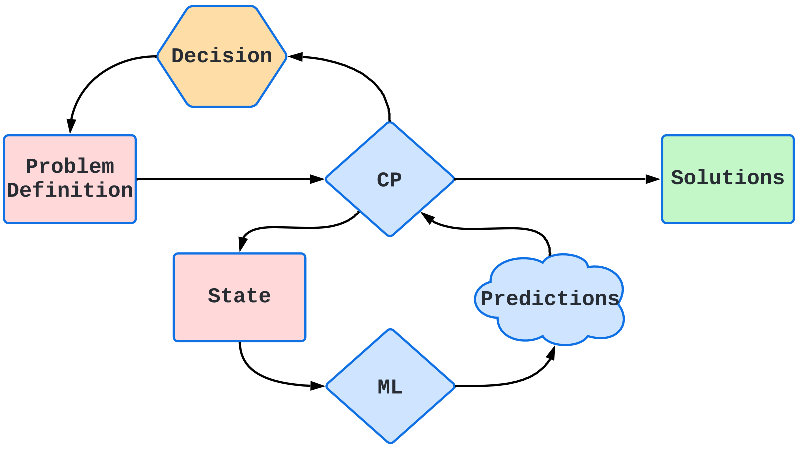

Figure 2 This scheme presents the integration of ML into CP performed by GenCP. It is freely inspired by Sec. 3.2.3 of Bengio et al.'s survey [2], which introduces an architecture for ML alongside Optimization Algorithms. The similarity is highlighted because the master algorithm (here, GenCP) repeatedly queries an ML model (here, an LLM) to obtain a prediction as a subroutine. In the context of this paper, the decision (search or propagation) has an impact on the problem definition (the CSP) because it may generate new variables, domains, or constraints during the solving process. The state is the current assignment of the variables.

3.1 New version of OTFS: GenCP

In traditional CP it is not common to generate new variables/domains/constraints during the search, while OTFS is based on this idea. OTFS begins with an empty or partially defined CSP (the CSP has less variables/domains/constraints than the CSP in traditional CP) and will generate variable/domains/constraints during the search for solutions.

GenCP is a new version of OTFS that makes two changes to the original version: **1)** the function that generates the domain *genD* calls an LLM to generate the domain of the current variable. **2)** Helping functions are added to represent implicit constraints.

Here is GenCP applied to text generation under constraints. An GenCP model can be defined as a pair of sets $\{\mathcal{M}, \mathcal{F}\}$, where:

- $\mathcal{M} = \{X, D, C\}$ represents the model of the problem.
- X represents the variables. The variables represent words.
- D represents the domain of the variables. A domain $d_i \in D$ contains a list of predicted words by an LLM.
- C represents the explicit constraints (constraints of the problem). A constraint $c_i \in C$ represents rules over text (e.g., number of words, number of characters, forbidden words, or symbols).
- $\mathcal{F} = \{G, B, H\}$ is a set of functions.
- G represents the set of generative functions: these functions explain to the solver how to generate variables/domains/constraints.
- B represents the set of Boolean functions: these functions tell the solver when a solution is found.
- H represents the set of helping functions: these functions are used to represent implicit constraints, for example ensuring that when a variable is generated, it helps obtaining a solution (to prevent the solver from attaining an infinite loop of generating variables).

3.1.1 Generative Functions

The set of generative functions $G = \{genV, genD, genC\}$ is such that:

- $genV$ generates a new variable with an empty domain and adds it to X.
- $genD$ calls the LLM with the current sentence formed by the model and sets the domain of the previously generated variable to the output.
- $genC$ generates the constraint(s) relevant to the current variables of the model to C. The constraints generated depend on the problem (e.g., generate a sentence that does not contain the letter “e”).

3.1.2 LLM integration

A variable is generated with an empty domain. To generate the domain of variables, $genD$ calls an LLM using $callLLM(sentence, parameters, k)$, where:

- $sentence$ is the current sentence represented by the variables of the model.
- $parameters$ represents sampling parameters ($top\_k, top\_p...$). For this paper, top_k is used exclusively for both GenCP and BS: the LLM answers k words ranked by probability, highest to lowest.
- k is the number of words asked to the LLM.

Since the parameters and k are not modified after the definition of the model, $callLLM(sentence, parameters, k)$ will be simply referred to as $callLLM(sentence)$.

3.1.3 Helping Functions

Helping functions represent implicit constraints, like avoiding infinite loops. In our current implementation, the set of functions H contains the following functions:

- H_o: it orders the domain of variables depending on the problem.
- $H_{onlyWords}$: it ensures that any word predicted by the LLM is a complete word and not the suffix or prefix of a word and it filters out any symbol or special character.

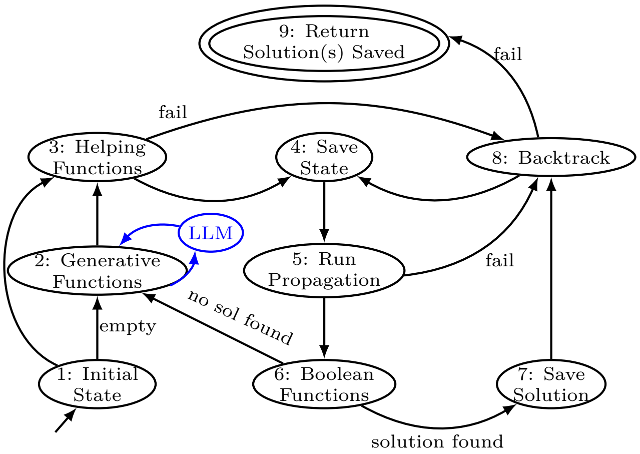

Figure 3 This graph illustrates the main steps in GenCP solving.

Algorithm 1 GenCP($\mathcal{M}, \mathcal{F}$), $\mathcal{M} = \{X, D, C\}$, $\mathcal{F}$ contains the generative and boolean functions.

```
1: $\mathcal{S} = \emptyset$; if $\mathcal{M}$ is not empty then go to 3.;
2: generativeFunctions($\mathcal{M}$);
3: helpingFunctions($\mathcal{M}$); if $\mathcal{M}$.X.$containsEmptyVariable()$ then go to 8.;
4: saveState($\mathcal{M}$);
5: propagation($\mathcal{M}$); if $\mathcal{M}$.X.$containsEmptyVariable()$ then go to 8.;
6: if not booleanFunctions($\mathcal{M}$) then go to 2.;
7: $\mathcal{S}$.add($\mathcal{M}$);
8: if backtrack($\mathcal{M}$) then go to 4.; else return $\mathcal{S}$;
```

3.1.4 Description of the new approach

The main steps of GenCP are depicted in Fig. 3 and Algorithm 1:

1. GenCP begins with an initial state. If the initial state is empty, the generative functions are called (**2.**), otherwise the helping functions are called (**3.**).
2. The generative functions *genV*/*genD*/*genC* are called (*genD* calls the LLM).
3. The helping functions are called to manage implicit constraints, backtracking if necessary (e.g., if the LLM generated an empty domain).
4. The current state of the model $\mathcal{M}$ is saved.
5. The propagation is called, if it fails the model backtracks (8.), else it calls the boolean functions (6.).
6. The Boolean functions are called to check if a solution has been found. If a solution is found, it is saved (7.) and the model backtracks (8.), otherwise the model calls the generative functions (2.).
7. The current sentence formed by the variables is saved as a solution.
8. GenCP backtracks to a previously saved state (4.) of the model and changes the choices made during propagation (5.). If no previous state was saved, then backtracking fails (9.).
   - When backtracking to a previously saved state, the model deletes all variables, their respective domains, and the constraints associated with them, that are not present in the previously saved state.
9. GenCP outputs the solution(s) that were saved or it indicates that no solution was found.

3.1.5 Enforce variability

Variability between two sentences is the number of words that are not equal at each position, for example:

- "The little boy is" and "The little cat is" have a variability of 1.
- "My name is John" and "John is my name" have a variability of 4.

To force a greater variability between solutions (greater than 2), a special backtrack called $backtrackTo(n)$ is used. Let the set of variables $X = \{x_1, \ldots, x_n, x_{n+1} \ldots, x_m\}$. The function $backtrackTo(n)$ deletes the variables x_{n+1} to x_m and causes a backtrack. For example, consider the sentence "I like to swim in the summer.". With $backtrackTo(2)$, "to swim in the summer." is deleted and the value of variable $x2 = $ "$like''$ is changed. The next solution might be "I want to break free.".

3.1.6 Ordering

For some tasks, not following the ordering strategies of the LLM (like top-k and top-p) can lead to better/faster solutions. Two other orderings are considered: PPL valuation and length of a word (depending on the average word length in the given language).

3.2 Modeling Example

Here is a simple example of how the search of GenCP works: for this paper the generative functions only generate variables one at a time but it is important to note that these functions can generate multiple variables, domains and constraints at once. Let us suppose GenCP has to generate a sentence beginning by "The" and containing between 10 and 15 words with exactly 60 characters. The following functions are needed:

- $currentSentence(\mathcal{M})$: outputs the current sentence the variables form.
- $callLLM(sentence)$: described in 3.1.2. Here k is equal to 10 (each time the LLM is called, it will output 10 words).
- $contains(sentence, word)$: outputs yes if the sentence contains the word and no otherwise.
- $nbChar(sentence)$: outputs the number of characters in the sentence.

The obtained model is $\{\mathcal{M}, \mathcal{F}\}$, where:

- $\mathcal{M} = \{X, D, C\}$:
- $X = \{x_1\}$.
- $D = \{d_1 = \{\text{“The”}\}\}$.
- $C = \emptyset$.
- $\mathcal{F} = \{G, B, H\}$.
- $G = \{genV, genD, genC\}$ is a set of functions, each function follows these steps:
  - generate $x_{|X|+1}$ and add it to X with an empty domain $d_{|X|+1}$.
  - $d_{|X|+1} = callLLM(currentSentence(\mathcal{M}))$.
  - $c_{remove_{over60char}((currentSentence(\mathcal{M}), d_{|X|+1})}$.
  - The constraints remove the words that make the current sentence exceed 60 characters from the domain of the current variable.
- $B = \{endNbWords, endNbCharacters, endLLM\}$ is a set of functions, each function behaves as follows:
  - $|X| >= 10 \wedge |X| <= 15$.
  - $nbChar(currentSentence(\mathcal{M})) == 60$.
  - $contains(callLLM(currentSentence(\mathcal{M})), \text{“}.''\text{)}$.

- $H = \{H_{ho}\}$:
  - $H_{ho} : order(d_{|X|+1})$.
  - To help attain the goal of 60 characters, the domain of the current variable is ordered such that before the 10th word the solver tries the longer words first and at the 10th word the solver tries the shorter words first.

With the above representation of the problem, GenCP is asked for 4 solutions, $backtrackTo(2)$ is used and the LLM is asked for 10 words maximum per call. The obtained solutions are:

1. The following is an article by the author of the above book.
2. The first time you see the movie version of your book on TV.
3. The New York Times has an article on the new book by Tim Wu.
4. The new year is here and we are ready to make the next step.

3.3 Illustrated Example

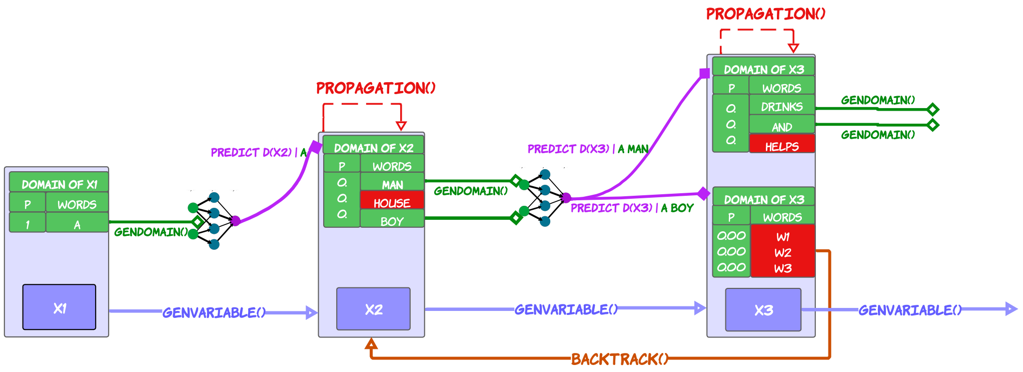

Figure 4 Illustrations of GenCP as a simplified framework with three variables and predictions of 3 values per LLM call, on a simple problem: generate a sentence that does not contain the letter e. For each variable, the predefined constraint c_i "the letter e is forbidden" is generated. A predefined domain with one word is defined for the first variable: A. The current sentence formed by the variable "A" is not a solution ($callLLM(“A'')$ does not answer a period ("."))$,$ so a new empty variable x_2 is generated. GenCP calls the LLM with the sentence "A" to predict the domain of x_2. The LLM model predicts three values: [man, house, boy]. c_2 is generated, causing the domain of x_2 to be filtered accordingly: house is removed. x_2 is then assigned to boy, GenCP generates the variable x_3 and calls the LLM with the sentence "A boy" to predict a new domain. Unfortunately, the domain of x_3 is empty, either because the LLM answered an empty domain or because this domain was entirely filtered during propagation. Hence, GenCP backtracks to x_2 and the value of x_2 is changed to man. In the same fashion as before, GenCP generates the variables x_3, and gives "A man" to the LLM that predicts: [drinks, and, helps]. c_3 is generated, filtering helps because it contains an e. The process continues until the domain of the next predicted variable contains a period (a solution is found) or the solver *fails*.

Fig. 4 illustrates GenCP as a simplified framework with three variables and predictions of 3 values per LLM call, on a simple problem: generate a sentence that does not contain the letter e.

4 Experiments

4.1 Experimental Conditions

4.1.1 Baseline

The experiments presented by Yao et al. are partially reproduced [40]. In particular, the constrained sentence generation tasks described in Tab. 1. Five LLMs were selected: GPT4, GPT4-O, Mistral Next, Claude 3.5, and Gemini. The four LLMs are prompted with the same example command given in [40]. For example, "Please create a sentence of exactly 82 characters." for the Sent-1 task[1]. Tab. 2 gives an overview of the performance of the five LLMs on the four tasks. The satisfaction rate is based on ten trials per task per model. In addition, Tab. 2 also shows that the LLMs perform well on the lexically constrained scenario task-4 with a 90+% satisfaction rate over ten trials. Also, as Yao et al. previously showed in their paper, LLMs struggle to produce constrained sentences involving counting (e.g., words and characters). They provide a nice picture of current LLM satisfaction capabilities by introducing new benchmarks. Unfortunately, the Yao et al. article only provides the benchmarks and some hints on reproducing them. However, it does not give any hints on how to solve the tasks associated with the benchmarks (see the original article for more details [40]).

Table 1 Four tasks on sentence generation used to compare BS and GenCP extract from [40].

name	words count	character count	lexical constraints
sent-1		$= 82$	
sent-2	$= 10$		$X_3 = \text{soft}$, $X_7 = \text{soft}$,$X_{10} = \text{math}$
sent-3	≥ 20	$\forall i, \lvert X_i \rvert \leq 6$	
sent-4			soft, beach, math

Table 2 Number of successes (#s), Number of fails (#f) and satisfaction rate (%sat) for each model (GPT-4, GPT-4.0, Mistral Next, Claude 3.5, Gemini) for each task (sent-1, sent-2, sent-3, sent-4).

name	GPT-4			GPT-4.0			Mistral Next			Claude3.5 Sonnet			Gemini		
	#s	#f	%sat	#s	#f	%sat	#s	#f	%sat	#s	#f	%sat	#s	#f	%sat
sent-1	1	9	10%	0	10	0%	0	10	0%	1	9	10%	0	10	0%
sent-2	0	10	0%	0	10	0%	0	10	0%	0	10	0%	0	10	0%
sent-3	1	9	10%	5	5	50%	0	10	0%	9	1	90%	1	9	10%
sent-4	9	1	90%	9	1	90%	10	0	100%	10	0	100%	1	9	10%

4.1.2 Hardware & Implementation

The experiments were performed on a laptop with Windows 10 Professional, 32 GB RAM, and Intel 16 CPU cores. The approach and the BS are implemented in Java 17 for easier comparisons.

[1] `https://chatgpt.com/share/b2834735-f7d8-468a-ba54-7da19dd6723c`

4.1.3 LLM choice

LLaMa [37] is responsible for the predictions of words as domains for the variables, mainly because an efficient implementation in C++ was recently released[2]. It allows running a model on a personal laptop and CPU (only) efficiently. Thanks to quantization [16] (model weight compression), the 7B parameters model (in Float16) of 13GB original size, in 4-bit quantization (Q4) drops to 3.9GB of RAM. However, the biggest model of LLama 65B (120GB), even in Q4, needs 38.5 GB of RAM. Thus, the LLaMa v1 model used in the experiments is LLaMa 7B Q4 with low temperature (i.e., ≤ 1, temp $= 0.8$).

When asked for k words, this version of LLaMa will take the same amount of time to ouput 1 word and 1000 words. To minimize the importance of k, *callLLM* outputs more than k words, a beam/variable only keeps k "valid" words. A "valid" word is a word that does not violate a constraint on its own. For example, a word that does not violate the constraint "does not contain the letter e".

4.1.4 Beam Search Technical Remarks

In the current implementations two halting conditions are defined for BS:

- First solution: when the current beam contains at least one solution, BS is stopped and output the solutions.
- All solutions: when the current beam contains at least one solution but another beam can continue to generate words without violating a constraint (for instance, it does not contain enough characters to satisfy a length constraint), the beam solutions are saved and BS continues with the remaining beams.

4.1.5 Benchmarks Settings

BS and GenCP are compared on some recent benchmarks described in Sec. 4.1.1.

To guarantee GenCP and BS to be judged on the generation of sentences of the same quality, a solution is a sentence that satisfies all the constraints of the current task and, when given this sentence, the LLM predicts a period ("."). Not to alter BS too much, words are ordered by probability (PPL is not used) and, since BS sentences have low variability, GenCP is used without $backtrackTo(n)$.

BS and GenCP are compared on the following criteria:

- Time in seconds.
- Number of solutions. GenCP was stopped when it found the same number of solutions as BS on a task. 0/1 means that BS found no solution while GenCP found one solution.
- The ratio solutions/outputs as a constraint satisfaction rate.
- For BS only, the number of bad outputs (number of outputs that are not a solution).
- For GenCP only, the number of backtracks.

4.2 Result Analysis

The results show that GenCP can be used to solve efficiently text generation under constraints problems. GenCP is faster than BS and all the outputs are solutions, contrary to BS where some outputs are not solutions.

[2] https://github.com/ggerganov/llama.cpp

Although the results suggest that GenCP succeeds in all tasks (see Tab. 3), it becomes particularly interesting when considering size constraints (e.g., sentences with a precise number of words or characters). It obtains sentences that satisfy the constraint with a low PPL score on sent-1 and sent-3 tasks.

GenCP also succeeds in producing sequences obeying lexical constraints in sent-2 and sent-4. However, the PPL and a human evaluation on these sentences show a substantial deterioration in term of quality (i.e., meaningfulness).

Therefore, regarding sent-1 and sent-3 tasks, GenCP is to be preferred, whereas for sent-4 and sent-2 tasks, LLMs prompted alone or joint with BS is still adequate.

Table 3 Comparison of BS and GenCP on the tasks of Tab. 1. Task considered (sent-i), Number of solutions (#sols), Time in seconds (s), Number of bad output (#badoutput), satisfaction rate (%sat) and Number of backtracks (#bk).

Experiments			BS			GenCP		
sent-i	k	#sols	s	#badoutput	%sat	s	%sat	#bk
1	5	1	108	9	10%	103	100%	45
	10	0	182	18	0%	177	100%	84
	20	1	399	58	1%	46	100%	13
	50	1	1123	109	≈ 0%	47	100%	13
2	5	5	34	0	100%	38	100%	38
	10	10	69	0	100%	36	100%	25
	20	20	140	0	100%	58	100%	40
	50	49	354	1	99%	134	100%	100
3	5	0	248	5	0%	36	100%	4
	10	2	510	8	20%	55	100%	6
	20	4	1030	16	20%	164	100%	38
	50	20	2633	30	66%	1174	100%	374
4	5	25	279	3	89%	308	100%	118
	10	30	513	8	78%	311	100%	114
	20	45	1123	14	76%	321	100%	104
	50	89	2928	40	68%	388	100%	27

Table 4 GenCP results for $k = 50$ when given approximately the same amount of time as BS in Tab 3. Number of solutions (#sols), Time in seconds (s), Memory usage in megabytes (MB), Number of backtracks (#bk).

Experiments			GenCP		
sent-i	k	#sols	s	MB	#bk
1	50	2	1123	136	79
2	50	355	222	208	222
3	50	488	2633	378	624
4	50	830	2929	676	680

4.2.1 Beam Search

BS and GenCP are compared in Tab. 3. In all tables, the number of backtracks is denoted by #bk. BS is slower than GenCP and has lower satisfaction rate (number of outputs that are solutions / total number of outputs), denoted by %sat. This is due to multiple facts:

1. Beam Search can not guarantee to find every solution.
2. Beam Search chooses the next word depending on the probability of the LLM.
3. At each step, BS considers k sentences, each sentence asks k words to the LLM, so each step considers k^2 words. BS orders these words decreasingly by probability and only keeps the k first.

Facts 2 and 3 explain why increasing k does not guarantee to find the same/more solutions, it might even cause BS to find less solutions.

Let us suppose $k = 5$, BS found one solution, and at depth 4, the candidate needed to find this solution was ranked 5 out of 25. Let us suppose now k is increased to 6: at each step BS will consider 36 candidates and take the 6 best ones. BS considers 11 more candidates than with $k = 5$; if at depth 4, the candidate needed to find the previous solution is now ranked 7 instead of 5, BS will not consider it and $k = 6$ will not find the solution found with $k = 5$.

4.2.2 GenCP

Table 5 Output sentences of GenCP on the experiments of Tab 1 associated with the task (sent-i), k, *backtrackTo* (*bkTo*), and Perplexity (PPL). In sent-4* a constraint was added so that "soft", "beach", "math" have to be separated by at least three words. Sentences with high perplexity were chosen to showcase the importance of low perplexity.

Experiments				GenCP
sent-i	k	*bkTo*(n)	PPL	sentence generated
1	50	NO	8	The following is an article by Dr David Hillon the subject of the role of prayer.
		2	13	The New York Times has an article on the new book by former President George Bush.
		3	6	The following information is taken from the website of the National Park Services.
2	50	NO	189	The following soft skills are required beach resort jobs math.
		2	169	The National soft drink association has beach balls and math.
		3	107	The most soft and comfortable of beach wear is math.
3	50	NO	8	The first time you see the movie The Big Short is like being hit by an ice cube in the face.
		2	5	The world is full of great ideas and the best way to get them out there is by using the power of the web.
		3	5	The first step in the right path is to know what you want and where you are going in life.
4*	50	NO	347	The following is an article by Dr math and science teacher beach high school in soft.
		2	593	The term of the contract is for math and science teachers beach to be able soft.
		3	48	The following data is based on the math and physics of beach waves and the soft sand.

Tab. 4 shows the capability of GenCP to generate more solutions than BS. GenCP is given the same time as BS for the same task and $k = 50$, GenCP obtains more solutions than BS. Note that for sent-1, without *backtrackTo* GenCP only obtains 2 solutions in 1123 seconds, while with *backtrackTo*(6) GenCP obtains 11 solutions in 1123 seconds.

The LLM-enhanced GenCP avoids the drawbacks of BS and proposes an alternative approach to text generation under constraints for the following reasons:

- GenCP can guarantee to find every solution (if any). Increasing k guarantees to find at least the same solutions previously found and potentially finds new solutions. Furthermore, it can offer more solutions than BS.
- All the outputs answered by GenCP are solutions (all the constraints are satisfied).
- GenCP offers more options for improvement, for example to ensure better variability (*backtrackTo* explained in 3.1.5 can be used) or other orderings than probability (3.1.6).

4.2.3 Variability and Perplexity

Tab. 5 demonstrates the importance of enforcing variability and perplexity. When GenCP generated solutions for Tab. 3 and 4, the maximum variability was 4. Tab. 5 shows that with $backtrackTo(2)/backtrackTo(3)$, sentences generated are almost completely different thanks to high variability (10+ for sent-3 for example).

Tab. 5 purposefully contains sentences with high perplexity to illustrate that this leads to a degradation in the sentence quality (i.e., low meaning).

All the sentences generated for sent-4 had the words “soft”, “beach” and “math” next to each other. To showcase the capability of GenCP to improve sentences, sent-4* was created: it is the same as sent-4 except that “soft”, “beach” and “math” must contain at least three words between them.

5 Discussion & Perspectives

5.1 GPU and CPU Interplay

The article shares a proof-of-concept showing that interesting results can be obtained using CPU resources combined with a small quantized LLM in a CP solver. However, LLMs, in general, work best with much larger computational resources and require GPU resources. Even though smaller models (e.g., Mistral 8x7B) sometimes manage to take top places in specific scenarios. The top spots in the LLM Elo rankings feature gigantic models [10]. Given their size, clusters of GPU are quickly mandatory. Hence, it would be interesting to study in more detail how the joint use of resources (for instance, CPU for solver and GPU for LLM) could improve the results of the paper and correspond to more real-world usage in industry.

5.2 Token Management

In this article, GenCP ignores tokens and works at the word level (pre-token). It is possible to handle tokens by adapting the problem modeling. Indeed, it is possible to consider a word as a meta-variable X_1 composed of several decision variables (e.g., X_{1_1}, X_{1_2}, X_{1_3}...). This is useful and straightforward, as it is not clear in advance how the tokenizer will cut the words. For instance, let us consider the following sentence: *The first step in the recruitment of a new hire is to make sure that the job requisition is clear.* Let us look at the assignments of the variables (space separates meta-variables, and semicolon decision variables): *The; first; step; in; the; rec;ruit;ment; of; a; new; h;ire; is; to; make; sure; that; the; job; requ;is;ition; is; clear;.* The word *recruitment* needs three decision variables because it is composed of three tokens (i.e., *rec, ruit* and *ment*). It is easy to manage in GenCP because it can generate as many variables as required. Nevertheless, the evolution of the CSP (generation of variables and domains) is rather technical and, therefore, depends on the tokenizer.

5.3 CSP Modeling

The idea that a CSP can evolve in response to external information is not new (e.g., Dynamic Constraint Network [12]). This dynamic vision of CSPs has been motivated by several real-world problems, particularly in product configuration [19]. GenCP proposes ML integration in modeling by letting LLMs manage operations for CSP domains during the resolution process. The “outside the world” information [3] is given by the LLM. The article shows that LLMs can contribute to CSP modeling for generation tasks. However, how ML/LLMs can be used for CSP modeling in general for any problem remains an open problem [14, 24, 38, 11].

6 Conclusion

This paper showed that combining CP solving of structural constraints and ML understanding of vague notions (like meaning) on the task of text generation under constraints obtains promising results. This paper presents GenCP, a new method that extends OTFS to make the domains manageable by LLM predictions. The results show that GenCP can generate meaningful sentences that ensure various properties like the number of words, number of characters, mandatory keywords, or some forbidden characters. The results also show that GenCP has 100% satisfaction rate and takes less time to output solutions of the same quality than a well-known technique in the field of text generation under constraints: Beam Search. GenCP provides multiple improvements thanks to ordering, enforcing variability and perplexity, allowing thus to obtain overall higher quality solutions than BS.

References

1 Andrea Bartolini, Michele Lombardi, Michela Milano, and Luca Benini. Neuron constraints to model complex real-world problems. In *Principles and Practice of Constraint Programming–CP 2011: 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011, Proceedings*, volume 6876, page 115. Springer, 2011.

2 Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021. doi:10.1016/j.ejor.2020.07.063.

3 Christian Bessière. Arc-consistency in dynamic constraint satisfaction problems. In Thomas L. Dean and Kathleen R. McKeown, editors, *Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim, CA, USA, July 14-19, 1991, Volume 1*, pages 221–226. AAAI Press / The MIT Press, 1991. URL: http://www.aaai.org/Library/AAAI/1991/aaai91-035.php.

4 Alexandre Bonlarron, Aurélie Calabrèse, Pierre Kornprobst, and Jean-Charles Régin. Constraints first: a new mdd-based model to generate sentences under constraints. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 1893–1901, 2023.

5 Alexandre Bonlarron and Jean-Charles Régin. Intertwining cp and nlp: The generation of unreasonably constrained sentences. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, 2024. To appear.

6 Alexandre Bonlarron and Jean-Charles Régin. Markov constraint as large language model surrogate. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, 2024. To appear.

7 Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, Jennifer C. Lai, and Robert L. Mercer. An estimate of an upper bound for the entropy of English. *Computational Linguistics*, 18(1):31–40, 1992. URL: https://aclanthology.org/J92-1002.

8 Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

9 Quentin Cappart, Thierry Moisan, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Andre A. Cire. Combining reinforcement learning and constraint programming for combinatorial optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(5):3677–3687, May 2021. doi:10.1609/aaai.v35i5.16484.

10 Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating llms by human preference, 2024. arXiv:2403.04132.

11 Parag Pravin Dakle, Serdar Kadıoğlu, Karthik Uppuluri, Regina Politi, Preethi Raghavan, SaiKrishna Rallabandi, and Ravisutha Srinivasamurthy. Ner4opt: Named entity recognition for ;optimization modelling from ;natural language. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 20th International Conference, CPAIOR 2023, Nice, France, May 29 –June 1, 2023, Proceedings*, pages 299–319, Berlin, Heidelberg, 2023. Springer-Verlag. doi:10.1007/978-3-031-33271-5_20.

12 Rina Dechter and Avi Dechter. Belief maintenance in dynamic constraint networks. In Howard E. Shrobe, Tom M. Mitchell, and Reid G. Smith, editors, *Proceedings of the 7th National Conference on Artificial Intelligence, St. Paul, MN, USA, August 21-26, 1988*, pages 37–42. AAAI Press / The MIT Press, 1988. URL: http://www.aaai.org/Library/AAAI/1988/aaai88-007.php.

13 Eugene C. Freuder. In pursuit of the holy grail. *Constraints*, 2(1):57–61, 1997. doi:10.1023/A:1009749006768.

14 Eugene C. Freuder. Conversational modeling for constraint satisfaction. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(20):22592–22597, March 2024. doi:10.1609/aaai.v38i20.30268.

15 Cristina Garbacea and Qiaozhu Mei. Why is constrained neural language generation particularly challenging? *arXiv preprint arXiv:2206.05395*, 2022.

16 Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *CoRR*, abs/2103.13630, 2021. arXiv:2103.13630.

17 Chris Hokamp and Qun Liu. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1535–1546, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi:10.18653/v1/P17-1141.

18 Nan Jiang, Maosen Zhang, Willem-Jan Van Hoeve, and Yexiang Xue. Constraint reasoning embedded structured prediction. *J. Mach. Learn. Res.*, 23(1), January 2022.

19 U Junker, F Rossi, P van Beek, and T Walsh. Handbook of constraint programming. *Chapter Configuration*, 2006.

20 Dan Jurafsky and James H. Martin. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition.* Pearson Prentice Hall, Upper Saddle River, N.J., 2009. URL: http://www.amazon.com/Speech-Language-Processing-2nd-Edition/dp/0131873210/ref=pd_bxgy_b_img_y.

21 Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), February 2016. doi:10.1609/aaai.v30i1.10080.

22 Elias B. Khalil, Bistra Dilkina, George L. Nemhauser, Shabbir Ahmed, and Yufen Shao. Learning to run heuristics in tree search. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 659–666, 2017. doi:10.24963/ijcai.2017/92.

23 James Kotary, Ferdinando Fioretto, Pascal van Hentenryck, and Bryan Wilder. End-to-end constrained optimization learning: A survey. In *30th International Joint Conference on Artificial Intelligence, IJCAI 2021*, pages 4475–4482. International Joint Conferences on Artificial Intelligence, 2021.

24 Connor Lawless, Jakob Schoeffer, Lindy Le, Kael Rowan, Shilad Sen, Cristina St. Hill, Jina Suh, and Bahareh Sarrafzadeh. "i want it that way": Enabling interactive decision support using large language models and constraint programming, 2024. arXiv:2312.06908.

25 Yixian Liu, Liwen Zhang, Wenjuan Han, Yue Zhang, and Kewei Tu. Constrained text generation with global guidance - case study on commongen. *CoRR*, abs/2103.07170, 2021. arXiv:2103.07170.

26 Michele Lombardi, Michela Milano, and Andrea Bartolini. Empirical decision model learning. *Artificial Intelligence*, 244:343–367, 2017. Combining Constraint Solving with Mining and Learning. doi:10.1016/j.artint.2016.01.005.

27 Ximing Lu, Sean Welleck, Peter West, Liwei Jiang, Jungo Kasai, Daniel Khashabi, Ronan Le Bras, Lianhui Qin, Youngjae Yu, Rowan Zellers, Noah A. Smith, and Yejin Choi. NeuroLogic A*esque decoding: Constrained text generation with lookahead heuristics. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 780–799, Seattle, United States, July 2022. Association for Computational Linguistics. doi:10.18653/v1/2022.naacl-main.57.

28 Mohsen Nafar and Michael Römer. Using clustering to strengthen decision diagram bounds for discrete optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(8):8082–8089, March 2024. doi:10.1609/aaai.v38i8.28647.

29 François Pachet and Pierre Roy. Markov constraints: Steerable generation of markov sequences. *Constraints*, 16(2):148–172, April 2011. doi:10.1007/s10601-010-9101-4.

30 Alexandre Papadopoulos, Pierre Roy, Jean-Charles Régin, and François Pachet. Generating all possible palindromes from ngram corpora. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 2489–2495. AAAI Press, 2015.

31 Guillaume Perez and Jean-Charles Régin. MDDs: Sampling and probability constraints. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, pages 226–242, 2017. doi:10.1007/978-3-319-66158-2_15.

32 Matt Post and David Vilar. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1314–1324, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi:10.18653/v1/N18-1119.

33 Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021. arXiv:2112.10752.

34 Florian Régin and Elisabetta De Maria. Using on-the-fly model checking to improve constraint programming for dynamic problems. In *2023 IEEE 35th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 393–398, 2023. doi:10.1109/ICTAI59109.2023.00063.

35 Jialin Song, ravi lanka, Yisong Yue, and Bistra Dilkina. A general large neighborhood search framework for solving integer linear programs. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20012–20023. Curran Associates, Inc., 2020. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/e769e03a9d329b2e864b4bf4ff54ff39-Paper.pdf.

36 Damien Sprockeels and Peter Van Roy. Expressing musical ideas with constraint programming using a model of tonal harmony. In *International Joint Conference on Artificial Intelligence*, 2024. To appear.

37 Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. cite arxiv:2302.13971. URL: http://arxiv.org/abs/2302.13971.

38 Dimos Tsouros, Hélène Verhaeghe, Serdar Kadıoğlu, and Tias Guns. Holy grail 2.0: From natural language to constraint models, 2023. arXiv:2308.01589.

39 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

40 Shunyu Yao, Howard Chen, Austin W. Hanjie, Runzhe Yang, and Karthik R Narasimhan. COLLIE: Systematic construction of constrained text generation tasks. In *The Twelfth International Conference on Learning Representations*, 2024. URL: https://openreview.net/forum?id=kxgSlyirUZ.

41 Ruibin Yuan, Hanfeng Lin, Yi Wang, Zeyue Tian, Shangda Wu, Tianhao Shen, Ge Zhang, Yuhang Wu, Cong Liu, Ziya Zhou, Ziyang Ma, Liumeng Xue, Ziyu Wang, Qin Liu, Tianyu Zheng, Yizhi Li, Yinghao Ma, Yiming Liang, Xiaowei Chi, Ruibo Liu, Zili Wang, Pengfei Li, Jingcheng Wu, Chenghua Lin, Qifeng Liu, Tao Jiang, Wenhao Huang, Wenhu Chen, Emmanouil Benetos, Jie Fu, Gus Xia, Roger Dannenberg, Wei Xue, Shiyin Kang, and Yike Guo. Chatmusician: Understanding and generating music intrinsically with llm, 2024. arXiv:2402.16153.