Survey: Controlling Language Models for Natural Language and Code

Sameer Pimparkhede, Pushpak Bhattacharyya
IIT Bombay
{sameerp,pb}@cse.iitb.ac.in

Abstract

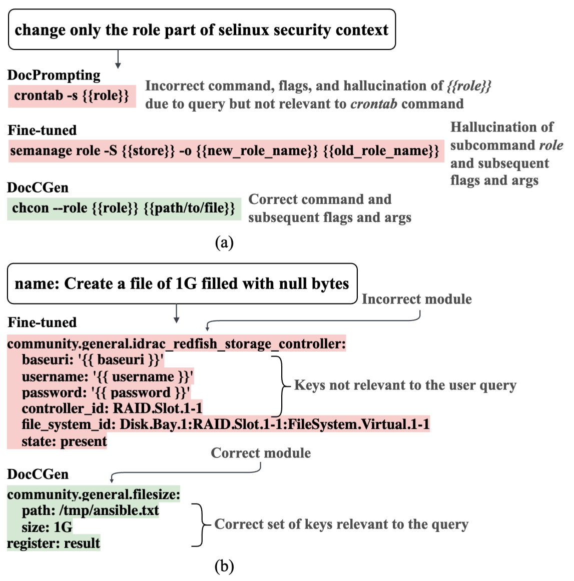
Large language models (LLMs) have achieved remarkable success across a wide range of natural language processing tasks, yet their outputs often remain uncontrolled—failing to adhere to explicit instructions, structural constraints, or user-specified requirements. This limitation is particularly critical in high-stakes applications such as code generation, structured summarization, and interactive systems, where correctness and constraint satisfaction are essential. In this survey, we present a comprehensive overview of decoding-time control methods for LLMs, focusing on three core settings: controlled generation for natural language tasks, program synthesis and domain-specific code generation, and lexically constrained decoding. We analyze representative techniques including schema-constrained decoding, static monitors, grammar-based control, and constrained beam search, highlighting their design principles, strengths, and trade-offs. We also review recent benchmarks such as InfoBench, FollowBench, and structured task suites—that provide fine-grained evaluations of constraint adherence. Despite growing progress, our analysis shows that current models struggle with multi-constraint satisfaction and compositional generalization. We conclude by discussing open challenges and future directions for building modular, reliable, and interpretable control mechanisms in large language generation systems.

1 Introduction

Large Language Models (LLMs) have emerged as powerful tools for a wide range of generative tasks such as summarization, machine translation, question answering, code generation, and open-domain dialogue. Their ability to produce fluent, coherent, and contextually relevant text has led to widespread adoption across academic and industrial applications. However, despite their strengths, these models often operate as black-box generators: they produce outputs based on statistical correlations in their training data without strong guarantees on factuality, safety, structure, or adherence to user intent. This lack of controllability limits the deployment of LLMs in scenarios where specific constraints must be satisfied—whether due to domain requirements, user preferences, or safety concerns.

Recent work has explored a range of control techniques across different problem settings. For instruction-following evaluation, InfoBench [14] proposes a fine-grained metric called the Decomposed Requirements Following Ratio (DRFR), which decomposes each instruction into multiple binary sub-requirements, allowing precise measurement of adherence across content, style, format, number, and linguistic constraints. In a complementary direction, FollowBench [7] introduces a benchmark for compositional and multi-layered constraint following, testing how models handle instructions that combine content, situation, style, format, and example-based constraints. Both studies show that even large models like GPT-4 often struggle when faced with simultaneous, structured constraints. Finally, [17] evaluate LLMs across five controlled generation tasks—including numerical planning, structured story generation, and constraint-preserving paraphrasing—revealing persistent gaps in models' ability to satisfy global semantic or structural conditions despite strong fluency.

In the context of code generation, constrained decoding techniques are increasingly being used to enforce structural correctness, execution validity, or schema adherence. For example, Execution-Guided Decoding [13] integrates real-time program execution into the decoding loop to eliminate semantically invalid completions. Other approaches like Retrieval-Augmented Code Generation [1] constrain model behavior using retrieved code snippets that serve as anchors for in-distribution output. In more structured enterprise settings, schema-guided


change only the role part of selinux security context
DocPrompting
Incorrect command, flags, and hallucination of *{{role}}* due to query but not relevant to *crontab* command
Fine-tuned
Hallucination of subcommand *role* and subsequent flags and args
DocCGen
Correct command and subsequent flags and args
(a)
name: Create a file of 1G filled with null bytes
Incorrect module
Fine-tuned
Keys not relevant to the user query
Correct module
DocCGen
Correct set of keys relevant to the query
(b)

```
crontab -s {{role}}
```

```
semanage role -S {{store}} -o {{new_role_name}} {{old_role_name}}
```

```
chcon --role {{role}} {{path/to/file}}
```

```
community.general.idrac_redfish_storage_controller:
   baseuri: '{{ baseuri }}'
   username: '{{ username }}'
   password: '{{ password }}'
   controller_id: RAID.Slot.1-1
   file_system_id: Disk.Bay.1:RAID.Slot.1-1:FileSystem.Virtual.1-1
   state: present
```

```
community.general.filesize:
   path: /tmp/ansible.txt
   size: 1G
register: result
```

Figure 1: Illustration of shortcomings with fine-tuning and DocPrompting [19] approaches with an example for (a) NL to Bash task (uses GPT Neo 1.3B) and (b) NL-to-YAML task (uses StarCoder2 3B) and the proposed DocCGen method to overcome the limitations.

decoding techniques have been proposed to ensure conformity to complex DSLs during code synthesis [12].

Meanwhile, lexical constraints—such as forcing the inclusion or exclusion of specific tokens—have been addressed using algorithmic decoding modifications. Constrained Beam Search [10] maintains beam candidates that satisfy prefix-based constraints at each step, allowing fine-grained lexical control. Grid Beam Search [2] generalizes this to support complex, multi-span constraints and diverse token placements without excessive beam expansion.

The need for controllable generation is especially critical in industrial and enterprise applications. In code generation, for instance, LLMs are increasingly used to assist with structured code synthesis in configuration files, APIs, and domain-specific languages. In such settings, generated outputs must conform strictly to schemas, typing rules, and toolchain compatibility. A minor violation of syntax or a hallucinated field can lead to system failure, security vulnerabilities, or downtime in production systems. Similarly, in enterprise-scale conversational agents and documentation assistants, factual inaccuracy or inappropriate phrasing may compromise trust, introduce legal liabilities, or fail to meet regulatory standards. Therefore, developing mechanisms to constrain, steer, and validate model outputs is not merely a research curiosity but a practical necessity for real-world deployment.

Despite these advances, enforcing constraints—particularly soft or abstract ones such as factuality, logical consistency, or user alignment—remains non-trivial. Hard constraints, such as output formats or lexical restrictions, are easier to impose through decoding-time interventions but can degrade fluency or diversity. Moreover, control strategies are often brittle or domain-specific, failing to generalize across tasks or scale with model size. Evaluation remains another open challenge: traditional generation metrics such as BLEU or ROUGE do not capture constraint satisfaction, and constraint-specific metrics often lack standardization or grounding in human preference.

This limitation in interpretability and transparency has broader implications beyond control. As AI systems become increasingly integrated into high-stakes domains—ranging from criminal justice to healthcare to financial decision-making—the need to understand why a model made a particular prediction becomes paramount. For instance, models used for intent classification in conversational AI must not only be accurate but also explainable, especially when user trust or error diagnosis is involved. In this context, recent work has proposed feature attribution techniques that map model decisions back to important input tokens—often using gradient-based or perturbation-based methods. These explanations are critical for identifying misleading patterns or ensuring that the model's reasoning aligns with human expectations. Moreover, in tasks like intent classification, explanations based on main verbs and semantic slots have shown promise in providing concise and meaningful rationales. By using curated datasets and hybrid techniques like the FRESH pipeline, researchers demonstrate how span-based explanations can support error analysis, improve model design, and foster greater transparency in LLM behavior.

To provide a unified perspective across these fragmented efforts, we present a comprehensive and technically detailed survey of controlled generation approaches, organizing the space into three core areas: natural language generation with factual and safety constraints, constrained code generation with structural and execution-oriented objectives, and lexical constraint enforcement using decoding-time methods. For each area, we identify the types of constraints involved, the specific control mechanisms proposed, and the trade-offs observed in empirical evaluations. Through detailed case studies, illustrative examples, and comparative tables, this survey aims to provide a unified understanding of the challenges and opportunities in controlled generation, laying the foundation for more reliable, verifiable, and purpose-aligned language technologies.

2 Controlled Generation for Natural Language Tasks

Natural language generation (NLG) has reached impressive fluency and generality with the rise of large language models (LLMs), yet this generative power comes with a lack of precise controllability. In many downstream applications—summarization, instruction following, paraphrasing, or story generation—practitioners often require models not just to produce relevant text, but to follow explicit structural or semantic constraints. Controlled generation in natural language settings is therefore an active research area, where the central goal is to ensure that outputs remain both faithful to the prompt and aligned with user-defined conditions, without compromising coherence or diversity.

A systematic investigation into this challenge is presented in the work of [17], who frame controlled generation as a capability that can be rigorously tested across diverse linguistic tasks. They benchmark several prominent LLMs—including GPT-3.5, GPT-4, and LLaMA 2—on a set of five challenging generation tasks designed to capture different control objectives. These tasks probe not only how well models can follow constraints, but also whether performance degrades under variation, ambiguity, or complexity in the prompt.

Numerical Planning: Here we test the ability of models to perform arithmetic reasoning or quantity tracking while generating a text plan. For instance, given a constraint like "Generate a travel plan that spans exactly 3 days," the model must ensure that the output describes precisely three day-wise activities, not more or fewer. Models often struggle with this form of global constraint satisfaction, particularly under open-ended instructions.

Content-Controlled Generation: Here we ask the model to include or exclude specified concepts. For example, the prompt may ask for a paragraph about environmental policy that includes "carbon tax" but avoids "renewable energy." Despite their strong generation skills, many LLMs fail to reliably satisfy such lexical and semantic constraints, especially when the instruction is long or indirectly phrased.

Story Generation: Here models are asked to produce narratives that obey a given high-level structure, such as beginning with a conflict and ending with a resolution. This task requires not only maintaining stylistic and structural consistency, but also enforcing temporal coherence—something LLMs often lose track of as generation length increases.

Rationale Generation: This requires a model to generate explanations that justify a decision or answer. For instance, given a multiple-choice question and a selected answer, the model must generate a plausible rationale that supports that answer based on the provided information. The challenge here lies in aligning reasoning chains with the target label, without hallucinating or introducing contradictions.

Controlled Paraphrase Generation It tests whether models can generate paraphrases that preserve meaning while satisfying constraints such as length limits, formality level, or mandatory keywords. This task combines semantic preservation with surface-level variation, and highlights how different models prioritize fluency versus control when trade-offs emerge.

[17] evaluate these tasks using both automatic metrics and human judgments, revealing significant variance in performance across model types and prompting strategies. Interestingly, while proprietary models like GPT-4 outperform open models on average, none of the models consistently satisfy all constraints across all tasks—indicating that current LLMs still lack robust mechanisms for generalized control. Moreover, few-shot prompting only modestly improves performance, suggesting that prompting alone is insufficient for certain types of control, such as global structure or multi-step reasoning.

Other than this, two recent benchmarks ,FollowBench and InfoBench provide comprehensive multi-dimensional evaluations across a wide variety of constraint types and generation settings.

FollowBench (Jiang et al., 2023) formulates the problem of controlled generation as the ability to satisfy multi-level fine-grained constraints embedded in prompts. It categorizes constraints along five types:

- Content constraints (e.g., “include the word ‘carbon tax’ but not ‘renewable energy’”),
- Stylistic constraints (e.g., “make it sound poetic”),
- Format constraints (e.g., “respond in bullet points”),
- Situational constraints (e.g., “you are a chef speaking to a child”),
- Example pattern constraints (e.g., “continue the dialogue as shown above”).

A key finding from FollowBench is that no single LLM consistently follows all constraint types, especially when multiple constraints are combined. GPT-4 performs best overall, yet even it struggles with format and content exclusion constraints under compositional prompts. Instruction tuning and in-context examples yield only partial improvements, suggesting the need for more robust decoding-level or planning-level mechanisms for fine-grained control. Complementing this, InfoBench (Qin et al., 2024) introduces a different yet synergistic methodology by proposing a new evaluation metric—Decomposed Requirements Following Ratio (DRFR)—to assess instruction adherence at a fine-grained, component level. Instead of holistic judgment or scalar scoring, DRFR decomposes a single instruction into multiple binary sub-requirements (e.g., “Does the output contain exactly 10 hotel reviews?”, “Is each review one sentence long?”). This decomposition enables precise error localization and better inter-annotator agreement compared to traditional direct scoring methods. InfoBench includes 500 instructions, decomposed into 2,250 sub-questions, and covers a rich set of constraints: Content, Linguistic, Style, Format, and Number, with more granular coverage and clarity than prior datasets. It also provides two levels of complexity—Easy and Hard sets—across 72 domains, including science, arts, marketing, and law.

3 Controlling Language Models for Code Generation

Language models are increasingly used to generate code across a wide range of domains—from general-purpose languages like Python and JavaScript to domain-specific languages (DSLs) for infrastructure-as-code, data pipelines, and configuration files. However, unlike natural language, code generation demands strict adherence to syntax, semantics, execution correctness, and often to external schemas or APIs. Even small deviations from expected output—such as a missing parameter, misused field, or mismatched type—can result in non-compiling code, runtime failures, or silent logic bugs. As such, controlling language model outputs in code generation is not just beneficial but essential for industrial deployment.

One principled method to enforce correctness is Constrained Semantic Decoding (CSD), introduced as part of the SYNCHROMESH framework [13]. SYNCHROMESH augments pre-trained LLMs with two key components: (1) Target Similarity Tuning (TST) for better example retrieval during prompting, and (2) CSD, which ensures syntactic and semantic validity of generated programs at decoding time without modifying the base model. In CSD, a domain-specific completion engine (CE) is used to determine the set of valid next tokens at each decoding step. The CE incorporates both context-free constraints (from the grammar) and context-sensitive rules (like type-checking or alias resolution). For example, in SQL generation, CSD ensures that column names conform to schema definitions and JOIN conditions respect alias scopes—errors that typical LLMs frequently make. CSD works with any target language, and is demonstrated over SQL, Vega-Lite (JSON-based data visualizations), and SMCalFlow (a Lisp-like task-oriented dialog language), showing significant improvements in both program validity (e.g., from 43% to 72% for GPT-3 in SQL) and execution accuracy. SYNCHROMESH’s modular constraint enforcement is a key advantage. Rather than relying on sampling-and-rejection or post hoc filtering, it dynamically restricts the model’s token selection to those that lead to valid programs. This alignment is computed via formal language derivatives (Brzozowski, 1964) and regular expression completions, enabling tight and efficient control. Experiments demonstrate that CSD and TST are complementary: TST steers the generation toward the correct conceptual structure, while CSD eliminates common implementation errors (e.g., type mismatches, invalid JSON structures, or SQL alias errors). In Vega-Lite, CSD prevents invalid chart encodings like "aggregate": "average" (where

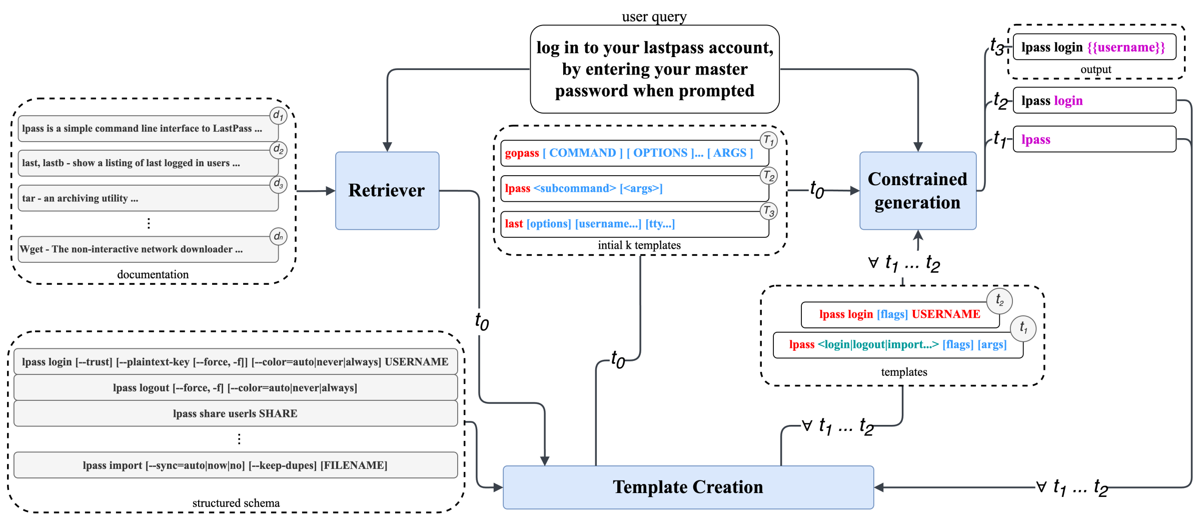

user query
log in to your lastpass account, by entering your master password when prompted
lpass is a simple command line interface to LastPass ...
last, lastb - show a listing of last logged in users ...
tar - an archiving utility ...
Wget - The non-interactive network downloader ...
documentation
Retriever
gopass [COMMAND] [OPTIONS]... [ARGS]
lpass <subcommand> [<args>]
last [options] [username...] [tty...]
intial k templates
Constrained generation
lpass login {{username}}
output
lpass login
lpass
lpass login [flags] USERNAME
lpass <login|logout|import...> [flags] [args]
templates
lpass login [--trust] [--plaintext-key [--force, -f]] [--color=auto|never|always] USERNAME
lpass logout [--force, -f] [--color=auto|never|always]
lpass share userls SHARE
lpass import [--sync=auto|now|no] [--keep-dupes] [FILENAME]
structured schema
Template Creation
t_0
$\forall\ t_1 \dots t_2$
t_1
t_2
t_3

Figure 2: Overview of DocCGen. For a given user query, top k relevant library documentations are retrieved and for which initial k templates are created. *Static* part of the template is shown in red, while the *variable* part is in blue. The variable field with a fixed position in the code is enclosed in angle brackets, for instance <subcommand>, as shown in the initial k templates block in the figure. The model is guided to follow one of the templates during decoding. Each time step t_i shows the step-by-step dynamic template evolution and constrained decoding output, adhering to the time-step template leading to the final generated code at $t3$.

"average" is not a valid keyword), correcting them to "mean" by construction.

While SYNCHROMESH ensures internal consistency within the generated code, Monitor-Guided Decoding (MGD) [1] takes a complementary approach by enforcing external consistency between the generated tokens and the evolving program state within a broader software repository. MGD introduces a decoding-time interface called a monitor, which integrates static analysis into the token selection process of an LLM. The goal is to ensure that code generation respects context-specific properties such as type constraints, class hierarchies, and variable scopes—particularly in statically typed languages. The core idea behind MGD is to intervene at trigger points during decoding, such as after a dot operator (.) or the invocation of an object method. At these points, the monitor uses the Language Server Protocol (LSP) to query a static analyzer for the set of valid continuations. For instance, if the object user is of type Customer, and the model generates user., the monitor intercepts and retrieves the list of fields and methods valid for Customer. The set of invalid next tokens is then masked from the output distribution, and only valid identifiers are allowed to be sampled. This form of control operates non-intrusively, requiring no model retraining or architectural changes, and can be applied to any decoder-based language model. The core idea behind MGD is to intervene at trigger points during decoding, such as after a dot operator (.) or the invocation of an object method. At these points, the monitor uses the Language Server Protocol (LSP) to query a static analyzer for the set of valid continuations. For instance, if the object user is of type Customer, and the model generates user., the monitor intercepts and retrieves the list of fields and methods valid for Customer. The set of invalid next tokens is then masked from the output distribution, and only valid identifiers are allowed to be sampled. This form of control operates non-intrusively, requiring no model retraining or architectural changes, and can be applied to any decoder-based language model.

A third form of control comes from domain-specific schema validation. [12] focuses on generating structured infrastructure-as-code (IaC) snippets such as Ansible YAMLs. These DSLs are brittle and constrained: fields must match strict schemas, including nested structure, required arguments, value types, and field ordering 2. The authors design a schema-aware constrained decoder, which uses JSON schema parsing to dynamically restrict the next valid tokens during generation. This decoding-time validator ensures that each field is syntactically valid and semantically adhere to the given task schema—without modifying model parameters or using fine-tuning. On a proprietary Ansible benchmark, this method reduced syntactic and semantic errors by over 50%, and required 70% fewer post-generation repairs 1.

A closely related yet broader solution is presented in Grammar-Constrained Decoding (GCD), introduced by [4] for structured NLP tasks. GCD frames output control as a grammar enforcement problem, where decoding is constrained by a context-free or regular grammar that encodes the valid output space. At each time step, a parser checks whether a partial sequence is derivable under the grammar, and prunes invalid continuations before sampling. The framework is model-agnostic and does not require finetuning, enabling structured generation in closed information extraction, constituency parsing, and structured slot-filling tasks. Although originally developed for NLP, this method maps naturally to code generation, where most target languages and DSLs already come with formal grammars or schemas. For instance, a JSON schema or BNF grammar for SQL or Terraform can be directly translated into a constraint engine in the GCD framework.

Taken together, these decoding-time constraint mechanisms reflect a spectrum of approaches for code control:

- **SYNCHROMESH** offers tight integration with task-specific semantics via typed completion engines.
- **Monitor-Guided Decoding** brings in repository-level awareness by coupling LLMs with static analyzers.
- **Schema-Constrained Decoding** targets structured DSLs where adherence to schema is essential.
- **Grammar-Constrained Decoding** provides a general-purpose symbolic layer for declarative control without finetuning.

These techniques are complementary. While grammar-based methods are language-agnostic and easy to implement, monitor-based methods offer greater semantic precision through program-state introspection. Schema-guided techniques are highly effective in structured formats like YAML or JSON, while completion engines like those in SYNCHROMESH enable domain-specific logic enforcement. Across all methods, the common theme is decoding-time modular control, which enables constraint satisfaction without sacrificing the generality or scalability of pre-trained language models. As code generation moves from prototype demos to enterprise deployment, these constraint-based generation techniques will become increasingly necessary to meet industrial expectations of correctness, safety, and usability.

4 Controlling language models on input side

ConCodeEval [8], a novel benchmark and study evaluating how well large language models (LLMs) adhere to and generate code constrained by structured schemas—critical for system-level programming in enterprise environments. Unlike prior work focused on natural language constraints, this study explores constraints represented directly as code in DSLs like JSON, YAML, XML, Python, and natural language. The authors propose two evaluation dimensions: **format efficacy**, which measures how input and output schema formats affect performance, and **constraint efficacy**, which assesses how well LLMs handle specific types and positions of constraints. Using a synthetic dataset of 602 schema samples across five representations, the paper evaluates several LLMs, including Granite, LLaMA, and CodeLLaMA, on two tasks: generating schema-compliant data 3 and validating whether given data adheres to schema constraints 4.

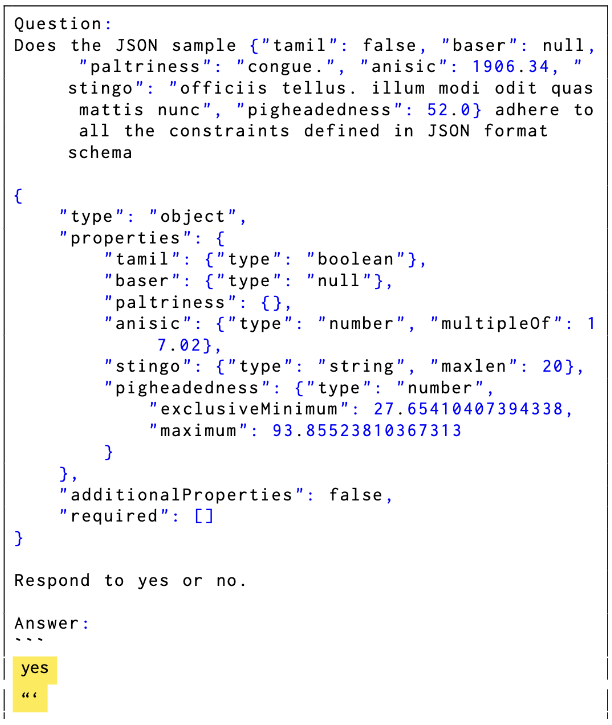
```
Write a JSON sample with field values as per the
     JSON format schema given below.

{
    "type": "array",
    "contains": {
        "type": "number",
        "multipleOf": 2.66,
        "exclusiveMinimum": 0.08231885995435284,
        "exclusiveMaximum": 5.1100233535478
    },
    "maxContains": 10
}

JSON sample:
```
[2.66, 5.22, 8.88, 11.54, 14.2]
``‘
```

Figure 3: The JSON sample generated (highlighted in yellow) by the Granite 20B model does not adhere to the *exclusiveMaximum* and *multipleOf* constraints specified in the schema.

The results show that LLMs perform best when constraints are presented in natural language for data generation but perform better with struc-
```



```
Question:
Does the JSON sample {"tamil": false, "baser": null,
      "paltriness": "congue.", "anisic": 1906.34, "
     stingo": "officiis tellus. illum modi odit quas
      mattis nunc", "pigheadedness": 52.0} adhere to
      all the constraints defined in JSON format
     schema

{
    "type": "object",
    "properties": {
        "tamil": {"type": "boolean"},
        "baser": {"type": "null"},
        "paltriness": {},
        "anisic": {"type": "number", "multipleOf": 1
             7.02},
        "stingo": {"type": "string", "maxlen": 20},
        "pigheadedness": {"type": "number",
            "exclusiveMinimum": 27.65410407394338,
            "maximum": 93.85523810367313
        }
    },
    "additionalProperties": false,
    "required": []
}

Respond to yes or no.

Answer:
```
yes
```
```

Figure 4: In the JSON sample, the values for fields *stingo* and *anisic* do not adhere to schema constraints. But the Granite 34B model gives the incorrect answer (highlighted in yellow) as *yes*.

tured formats like JSON or YAML for validation. Surprisingly, Python schemas performed worse, likely due to interference from the model's general-purpose code generation tendencies. Additionally, constraints located in the middle or beginning of schema contexts are more frequently missed, emphasizing the impact of token position in LLM comprehension. Models struggle particularly with numerical constraints such as ‘multipleOf‘ or ‘exclusiveMinimum‘, suggesting token-level and training limitations. The study recommends placing critical constraints at the end of schema definitions and highlights JSON and YAML as preferred formats for enterprise applications. Overall, the paper advocates for better schema comprehension methods and control strategies (e.g., constrained decoding) to improve LLM utility in system-level code generation.

5 Lexically Constrained Generation

In many sequence generation scenarios, users or downstream systems may require that certain words or phrases *must appear* (or be excluded) in the output. This requirement arises naturally in a variety of tasks including machine translation, summarization, image captioning, and controlled text generation. Known as **lexically constrained generation**, this task challenges language models to maintain fluency and relevance while satisfying **hard constraints** on the output vocabulary. Unlike soft prompts or fine-tuning, which provide probabilistic steering, lexically constrained decoding allows for **explicit, rule-based guarantees**—an essential property for interactive and safety-critical applications.

A seminal approach to this problem is **Constrained Beam Search (CBS)** [**?**]. CBS modifies the decoding process by tracking constraint satisfaction alongside the standard beam search. Let $C = \{c_1, c_2, \ldots, c_k\}$ be a set of required phrases. During generation, CBS partitions the beam based on which subset of C has been satisfied so far. At each decoding step t, the model selects the top-B hypotheses from extended beams, considering only those that:

- match a prefix of an unsatisfied constraint,
- complete a constraint,
- or proceed fluently if all constraints are already satisfied.

Formally, let $y_{1:t}$ be a partial sequence and $\mathcal{S}(y_{1:t}) \subseteq C$ the set of satisfied constraints. CBS maximizes:

$$\arg\max_{y_{1:T}} \log p(y_{1:T} \mid x) \quad \text{s.t.} \quad \mathcal{S}(y_{1:T}) = C$$

That is, only sequences that satisfy all constraints are considered valid.

While CBS offers control guarantees, it suffers from scalability issues as the number of constraint subsets grows. To address this, **Grid Beam Search (GBS)** [**?**] generalizes CBS by organizing beam hypotheses into a lattice over the power set of C. Each beam cell corresponds to a constraint subset $S \subseteq C$ and stores the top hypotheses satisfying S. At every step, beams are updated by:

1. extending hypotheses in B_S with next tokens,
2. moving them into $B_{S'}$ if new constraints are satisfied.

GBS enables efficient constraint tracking and supports unordered, overlapping, or partial constraints. It was originally applied to **open-vocabulary image captioning**, where object detection models produce keywords (e.g., *zebra*, *skateboard*) that must appear in the generated captions.

More broadly, lexically constrained decoding has enabled a wide range of applications:

- *Interactive Machine Translation*, where domain-specific terms must appear in the output;
- *Controlled Summarization*, preserving named entities or sentiment markers;
- *Image Captioning*, with visual objects enforced as output phrases;
- *Safety Enforcement*, where harmful completions are explicitly blocked.

Despite its utility, hard-constrained decoding introduces trade-offs:

- **Search overhead**: the beam size grows with 2^k subsets of k constraints;
- **Rigidity**: it cannot handle synonyms or fuzzy matches;
- **Reduced diversity**: hard constraints may hinder natural phrasing.

Recent works explore hybrid methods that blend lexical constraints with soft penalties, sampling, or post-editing, but these remain less stable and require additional tuning.

In sum, lexically constrained decoding offers a powerful mechanism for explicit control. Unlike prompt engineering or model tuning, it provides interpretable, enforceable constraints suitable for high-precision generation. As LLMs are increasingly deployed in structured applications, these techniques form a critical part of the decoding-time control toolkit.

6 Explainability in intent classification

Intent classification is done by 4 machine learning methods. Two of them are statistical machine learning methods, namely Support Vector Machine and Random Forest algorithms. Other two are deep neural network architectures . A simple feed-forward neural network consisting of multiple layers and Long-Short Term Memory (LSTM) based neural networks is used for this classification task .

Raw dataset has been cleaned by typical preprocessing steps for text sequence data . This is done by converting data to lowercase, removing punctuation, and special characters. Trivial stop words like "is" and "the" are not removed to experiment on the performance of explainability techniques and check their robustness. Words in the text sequences are converted to vectors of 300 dimensions by using word embeddings. For other experiments, word embeddings are not used, but words are converted using categorical feature encoding, which maps every word to a unique number. Finally, every sequence is padded at the end with blank space characters to convert all samples to an equal length of 30.

Testing dataset has been improved by doing manual perturbation of the text sequences. Perturbation method followed is replacing nouns and adjectives by their synonyms or antonyms. It has been ensured that the syntax and grammar of the text sample are intact while creating perturbation.

Support vector machine is a statistical machine learning approach that classifies the points by separating them with a linear or non-linear decision boundary and simultaneously ensuring that the decision boundary is at maximum length from support vectors to increase robustness of classification and better generalization. Support vector machine can be divided into the Hard margin and soft margin approaches. Soft margin SVM allows one to allow misclassification and better generalize the model, especially when samples are not linearly separable. Here we have used a soft margin Support vector machine approach that uses fitting a polynomial kernel of degree 3 to best generalize the model.

Another statistical machine learning technique used is randomForest, which is one of the best and common classification techniques. It is a bagging technique to assemble a number of decision trees. Decision trees classify the given sample based on a series of questions based on multiple features. This separates the samples by non linear decision boundary. But a decision tree has a risk of overfitting the data. Hence, the model ensembling technique is used to combine multiple trees and divide the data among them. This is done by the bagging technique, and as a result, we get the randomForest classifiers.

Feedforward neural network architecture consists of 2 deep neural layers, each consisting of 150 neurons and 100 neurons, with ReLU activation. It is followed by a dense layer with 8 neurons with a softmax activation function.

LSTM-based neural network consists of two LSTM layers of 64 and 32 neurons each, with a recurrent dropout layer and ReLu activation function. Similar to a FeedForward neural net, it is followed by a dense layer of 8 neurons and a softmax activation function.

These four models have been combined with 4 explainability techniques . Two of them are perturbation-based, namely LIME and Anchors; the other two are gradient-based techniques, which are Layer-wise Relevance Propagation.

LIME is a perturbation-based technique that explains the decisions of the model locally on a particular example. It is often impossible for an explanation to be completely faithful globally unless it describes the model itself but for an explanation to be meaningful it must be locally faithful. This means it should be precise in terms of explaining the feature importance for a particular sample. The features that are important locally may not be that important globally. Lime is a model-agnostic technique.LIME define an explanation as a model g G, where G is a class of potentially interpretable models, such as linear models, decision trees etc. It is ensured that model G should should be simple in terms of complexity. This can be done by limiting the depth of the decision tree or using a small number of features in the linear regression model. LIME creates a perturbation around the sample locally by masking some of the words/features in the text sequence. The perturbation samples created are at least a distance from the original sample.

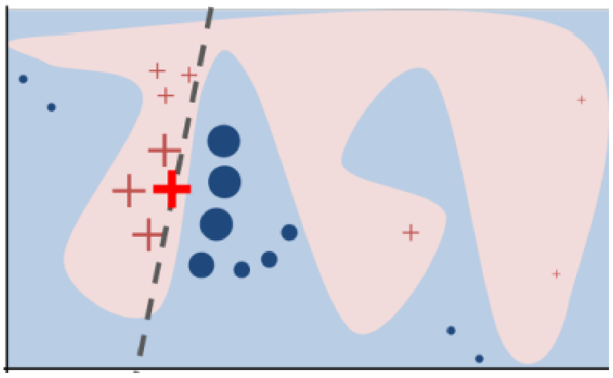


Figure 5

$$\xi(x) = \underset{g}{\operatorname{argmax}} L(f, g, \pi_x) + \Omega(g) \qquad (1)$$

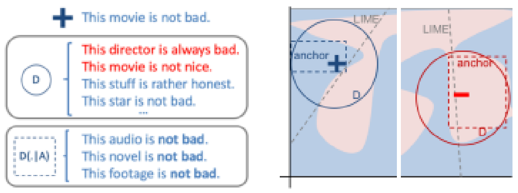
L(f, g, π) is a measure of how unfaithful g is in approximating f in the locality defined by π_x.Ω(g) is complexity of the linear model g which is be approximation of our original model.So the equation suggest that we have to increase the interpretability of faithfulness of model with the restriction that complexity of local model should be minimized. [15]

$$L(f, g, \pi_x) = \sum_{z,z'} ((f(z) - f(z'))^2 \pi_z \qquad (2)$$

π weighs the sample according to it's proximity with original sample. This ensures to penalize the perturbation for from sample.

Even if the Linear explanations by LIME are local it's coverage is not clear. Unclear coverage can lead to low human precision, as users may think an insight from an explanation applies to unseen instances even when it does not. For example in the task of sentiment analysis of movie reviews , "The movie is not bad" represents positive review and the "The movie is not good" negative review. The coverage of the word "not" is not clear here unless it is combined with the phrases "bad" and "good". This problem is solved by another model agnostic technique which is based on if-then rules called anchors.In other words, For the sample on which the anchor holds, the prediction is always the same. The explanation given by anchors in the above example is "not bad" and "not good" instead of just the word "not". [15]

If we want to explain the prediction f(x) of sample x to users then anchors uses following method to explain prediction to user. Let A be a rule (set of predicates) acting on such an interpretable representation, such that A(x) returns 1 if all its feature predicates are true for instance x.Let D(·|A) denote the conditional distribution when the rule A applies A is an anchor if A(x) = 1 and A is a sufficient condition for f(x) with high probability.If a sample z from D(z|A) is likely predicted as Positive (i.e. f(x) = f(z)) for all the perturbation of the samples then rule A is stated as sufficient condition for explaining the decision of model.


This movie is not bad.

This director is always bad.
This movie is not nice.
This stuff is rather honest.
This star is not bad.
...

This audio is not bad.
This novel is not bad.
This footage is not bad.

Figure 6: Difference between LIME and anchors prediction

Creating perturbations for techniques like LIME and Anchors is challenging task when it comes in the context of text sequence or language tasks. Adding noise in the text data should not change the meaning of the sentences and should be grammatically correct. This is achieved using various tricks.Nouns and adjectives in the text are replaced by there synonyms which are drawn from wordnet or word embeddings like word2vec. [6] Words can also be replaced with other words similar POS tags. In some cases words are replaced by creating small typo's in it like "good" is replaced with "god" to make model and explainability techniques more robust. Many time in the sentiment analysis task some words play positive role in some samples in others. For example the movie review of a funny movie "The movie is was really funny and dramatic" represents positive review but at the same time is same review is given to tragic movie then it represents negative review. [3] Words which play roles in different labels of reviews can be used to create perturbations. Here intent classification text are changed with adding words which are important for other tags. [16] For example "Which flight is from Mumbai to Pune" represents intent of "ATIS Flight" and "what is cheapest rate of fight from mumbai to banglore" represents intent "ATIS Airfare" important words from both examples are added to get the sample "What is the cheapest flight form mumbai to banglore" which represents tag as "ATIS flight" but model gets confused between words cheapest and flights.

Perturbation based techniques are good at explaining the model decisions locally but they depend on quality of perturbations and do not explore the gradient of the models in deep neural networks. Gradient based explainability techniques such as Layer wise relevance propagation (LRP) , Integrated gradients exploit that aspect and provides attribution scores by calculating gradient of output with respect to input feature which can be used to calculate attribution scores for input. [11]

in LRP relevance score of neuron k is distributed in lower level neurons like neuron j using the rule given below.

$$R_j = \sum_k \frac{z_{jk}}{\sum_j z_{jk}} R_k \tag{3}$$

Numerator term z_{jk} acts as a measure of how much the contribution of neuron j in forward propagation and hence in decision making. Denominator ensures that attribution score of higher-level neurons is conserved while distribution. Hence total attribution score in the network remains conserved and the attribution scores of the input is equal to attribution score of the output.

The gradients are generally noisy in the middle layer and there is a need to emphasize on different aspects in different layers and variations of basic LRP rules are applied throughout the network. These rules are listed below.

1. Base LRP rule (LRP-0):

$$R_j = \sum_k \frac{a_j w_{jk}}{\sum_{0,j} a_j w_{jk}} R_k \tag{4}$$

   - This rule distributes the relevance scores of each neuron in its lower layers using the conservation property.
   - If the activation is zero or the edge weight for some neuron, is zero then, the relevance of neuron Rj will evaluate to zero. This is correct because if weight is zero for particular neurons then it's not contributing to output.
   - LRP-0 rule is applied at the output layer and relevance for outermost layer neurons is activation value stored in it.

2. Epsilon Rule (LRP-ϵ):

$$R_j = \sum_k \frac{a_j w_{jk}}{\epsilon + \sum_{0,j} a_j w_{jk}} R_k \tag{5}$$

   - Gradients in the middle hidden layers are noisy and would tend to suppress important features.
   - To solve this problem small term ϵ is added in the denominator. The role of ϵ is to absorb some relevance when the contributions to the activation of neuron k are weak. This leads to sparse and less noisy explanations.

3. Gamma Rule (LRP-γ):

$$R_j = \sum_k \frac{a_j \cdot \left(w_{jk} + \gamma w_{jk}^{+}\right)}{\sum_{0,j} a_j \cdot \left(w_{jk} + \gamma w_{jk}^{+}\right)} R_k \tag{6}$$

   - For the input layer LRP-γ rule is applied to highlight positive features over negative features.

- This helps in looking for explanations more smoothly and hence easy to interpret for humans. The contribution of positive features can be controlled by tuning the hyperparameter.γ

Many of the gradient-based techniques do not follow important axioms sensitivity and invariance. [18]

sensitivity : An attribution method satisfies Sensitivity(a) if for every input and baseline that differ in one feature but have different predictions, the differing feature should be given a non-zero attribution. [18]

LRP is easy to implement and can be applied to any type of data which makes it flexible. Due to this flexibility, LRP can be used in a wide variety of tasks including text, images, or tabular data. Using the combination of LRP rules, good-quality explanations are obtained in the output. Explanations provided by LRP are stable and hence can be trusted. LRP is also computationally less expensive compared to perturbation-based techniques like LIME and anchors.

Here Deep Neural network is represented as a function $F \in [0, 1]$. Input is represented as a list of tokens $x = (x_1, x_2, x_3.....x_n) \in R^n$. The baseline input vector is represented as x' and attribution scores relative to the baseline for each token are represented as $a_1, a_2, a_3....a_n$ each a_i is the importance of token x_i for the model's decision.

Ignorance towards sensitivity causes the explainability technique to focus on non-important features.LRP tackle the Sensitivity issue by employing a baseline and in some sense try to compute gradients by including terms from the baseline in the numerator and denominator instead of normal gradients at the input but this approach breaks the invariance axiom. [18]

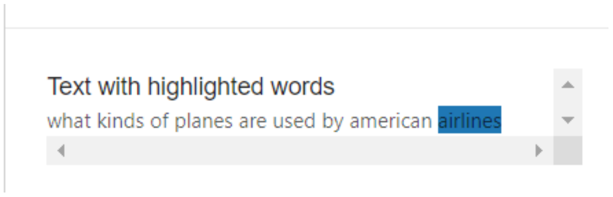
Implementation invariance: Two networks are functionally equivalent if their outputs are equal for all inputs, despite having very different implementations. These axioms suggest that attributions for input features should depend on only the gradient of output with respect to the input.

Integrated Gradient combines the Implementation invariance of gradients along with the Sensitivity of techniques like LRP or DeepLift. We consider the straightline path from the baseline x' to the input x, and compute the gradients at all points along the path. Integrated gradients are obtained by cumulating these gradients. Specifically, integrated gradients are defined as the path integral of the gradients along the straightline path from the baseline x' to the input x. Baseline vector selected for the intent classification is a zero embedding vector. Integrated gradients aggregate the gradients along the inputs that fall on the straight line between the baseline and the input. This allows this method to compute attributions efficiently as compared to other path gradient methods which calculate gradients across multiple paths and average out them at the end. [18] Expression for integrated gradients is as follows

$$IG_i(x) = (x - x') \sum_{k=1}^{m} \frac{\partial f(x' + \frac{k}{m}(x - x'))}{\partial x_i} \quad (7)$$

$$\sum_{i=1}^{n} IntegratedGrads_i(x) = F(x) - F(x') \quad (8)$$

Equation (3) denotes practical implementation of integrated gradients where m is the number of steps instead of integration. Here the number of steps have been taken as 50. Later integrated gradient computed for each feature Xi is added to obtain to get the difference attribution score for sample x for model and x' for baseline model. Integrated gradients satisfy completeness axiom which was states that attributions add up to difference between the output of F at the input x and the baseline x'. This was stated by the authors of LRP as requirement for explainability technique. [18] [11]

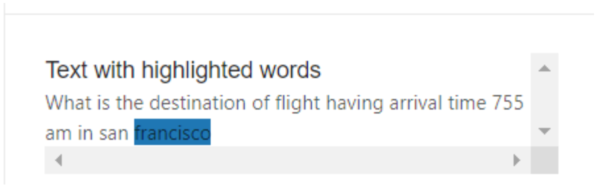

Text with highlighted words

what kinds of planes are used by american airlines

Figure 7: Explanation by anchors on sentence 'What is the destination of flight having arrival time 755 am in san francisco'

All the techniques like integrated gradients, LRP, LIME, and anchors are not required to be implemented on the training routine. Specifically Integrated gradient is easy to implement and here it is implemented directly on the trained deep neural network-based model using tensorflow/Pythorch libraries.

But these post-hoc explanations are often not reliable as these are third-party techniques and are



Text with highlighted words

What is the destination of flight having arrival time 755 am in san francisco

Figure 8: Explanation by anchors on sentence 'what is the destination of flights having arrival time 755 am in san francisco

often noisy. Hence it is stated that it is better that explanation comes through the model itself after is trained. Attention is one such method but it lacks faithfulness and there has been quite a debate about whether attention can be used as an explanation or not. Hence there are other technique stated by [9] which generates rationale in an unsupervised manner by changing objective function. It also help predictability at the same time using REINFORCE method.

Method by [9] involves training a neural model to make predictions, and then using a separate neural network to generate rationales for the predictions. The rationale generation network takes the input features and the predicted output as inputs, and produces a set of binary masks that indicate which input features were important for the prediction.

This technique contains two models generator and an encoder which are trained jointly to help each other's prediction. The generator finds rationale with maximum likelihood from a given piece of text and it's output is given to the encoder for classification. Rationale extraction is a completely unsupervised process.[9] Ground truth rationals are never provided to model during training. Rational extraction is done following two conditions, First, the rationales should be short and coherent and the second condition is it should be faithful for the predicted label.[9] The generator is a sequence tagging model where each word/feature is tagged to be rational or not. This done using REINFORCE methodology.

for given input sequence of length $(x_1, x2......xt)$ generator predicts $(z_1, z_2........zt)$ where $z \in [0, 1]$ which is decison for every word for being rational. The predicted rational set is denoted by (z, x) and it is given to the encoder. prediction of each word is treated as independent of each other in the generator. The generator is a simple bidirectional RCNN model with a softmax layer to give a probability distribution of tags for each word.[9]

Predicted rationale (z, x) is given to the encoder model which encodes it and outputs the resultant vector $enc(z, x)$ After this generator and encoder are trained jointly to improve each other's predictions. [9]

To follow the condition of the faithfulness of explanation encoder is trained with the objective function given by equation 6. This ensures that the predicted label is closer to the gold label even if rationales are given instead of complete input text. The second condition specifies that explanations must be short and continuous. The first term in equation 6 penalize long length rationales. The second term ensures that rationales are continuous by penalizing the long distance between predicted rationale. Final training is done to minimize the cost function expected value of cost function $L(z, x, y) + \Omega(z)$. Encoder model is chosen as RCNN as it is better to get continuous n-gram than RNN or LSTM. [9]

for given input sequence of length $(x_1, x2......xt)$ generator predicts $(z_1, z_2........zt)$ where $z \in [0, 1]$ which is decison for every word for being rational. The predicted rational set is denoted by (z, x) and it is given to the encoder. prediction of each word is treated as independent of each other in the generator. The generator is a simple bidirectional RCNN model with a softmax layer to give a probability distribution of tags for each word.

$$L(z, x, y) = ||enc(z, x) - y||^2 \tag{9}$$

$$\Omega(z) = \lambda_1||z|| + \lambda_2 \sum_t |z_t - z_{t-1}| \tag{10}$$

But REINFORCE method involves a lot of hyperparameters. Hence it becomes very costly to experiment with these models. This also has a lot of variance and produces unstable results. Many times [9] produces complete input text or blank rationale as well which doesn't give any useful information. To solve this problem [5] propose to decompose the generator and encoder model and train them independently. This method is stated as Faithful Rationale Extraction from Saliency thresholding (FRESH). [5]

In FRESH, the generator generates the rationale using heuristics that come from any post-hoc explanation techniques which need not be faithful. Later it trains an extractor model which performs a task similar to part of speech tagging to assign a binary mask to each token in the text sequence.

FRESH divides its approach into 3 stages. In the first stage, a classifier model is trained on the complete input text model. Then its rationale is constructed using any standard post-hoc explanation methods like LIME, Attention or gradient-based approaches. These explanations need not be faithful. Rationale from the generator model is given to the extractor model which performs the task to sequence labeling on input data using heuristics given by the generator. This is termed as the extractor model. It assigns binary tags to each token. hence we have a corresponding 1/0 sequence for each text sample. The third stage is again a classifier model. It is trained on the output of the extractor model. This output is set of rationale and not a complete sentence. Hence it can be easily tested whether the model's explanations are faithful or not. Here generator and encoder are entirely disconnected and independent unlike [9]. This produces more stable explanations and also lowers the number of hyperparameters to be tuned. [5]

7 Summary, Conclusion and Future Work

This survey has reviewed recent advances in decoding-time control of large language models across natural language tasks, code generation, and lexically constrained generation. We examined approaches that inject symbolic constraints, schema validation, static analysis, and grammar-based control directly into the decoding process, allowing precise intervention without modifying model parameters. These methods enable more reliable adherence to task-specific requirements, ranging from field-level correctness in structured code to keyword inclusion and instruction following in natural language. Despite this progress, current models often fall short when confronted with multi-constraint, compositional, or semantically rich prompts. Evaluations on benchmarks such as InfoBench and FollowBench demonstrate that even the strongest models struggle to balance fluency with strict constraint satisfaction, particularly in scenarios requiring numeric planning, stylistic control, or structural alignment. Looking ahead, future research will benefit from combining symbolic control mechanisms with learned representations, allowing models to adapt dynamically to evolving constraints during generation. Developing unified decoding frameworks that can incorporate various control signals—while maintaining generality—remains an open challenge. Additionally, interpretable and task-specific evaluation metrics will be essential to track fine-grained control progress, especially as models are deployed in safety-critical and domain-sensitive settings. Controlled generation is becoming a necessary component of reliable language model deployment. As applications grow more complex, decoding-time control offers a scalable and modular path to bridge the gap between general-purpose modeling and precise, user-aligned generation.

8 acknowledgement

I would like to offer my sincere gratitude to Dr.Pushpak Bhattacharyya, my RnD guide, without whose guidance this report wouldn't have been possible. I would also like to thank all the members from Gupshup technology for their support and valuable feedback.Finally I would like to thanks members of CFILT lab, for their support throughout the entire duration of my RnD.

References

[1] Lakshya A Agrawal, Aditya Kanade, Navin Goyal, Shuvendu K Lahiri, and Sriram K Rajamani. Guiding language models of code with global context using monitors. *arXiv preprint arXiv:2306.10763*, 2023.

[2] Guanhua Chen, Yun Chen, Yong Wang, and Victor O.K. Li. Lexical-constraint-aware neural machine translation via data augmentation. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 3587–3593. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Main track.

[3] Gamaleldin Elsayed, Shreya Shankar, Brian Cheung, Nicolas Papernot, Alexey Kurakin, Ian Goodfellow, and Jascha Sohl-Dickstein. Adversarial examples that fool both computer vision and time-limited humans. *Advances in neural information processing systems*, 31, 2018.

[4] Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured nlp tasks without finetuning. *arXiv preprint arXiv:2305.13971*, 2023.

[5] Sarthak Jain, Sarah Wiegreffe, Yuval Pinter, and Byron C. Wallace. Learning to faithfully rationalize

by construction. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4459–4473, Online, July 2020. Association for Computational Linguistics.

[6] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328*, 2017.

[7] Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. FollowBench: A multi-level fine-grained constraints following benchmark for large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4667–4688, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[8] Mehant Kammakomati, Sameer Pimparkhede, Srikanth Tamilselvam, Prince Kumar, and Pushpak Bhattacharyya. Concodeeval: Evaluating large language models for code constraints in domain-specific languages. *arXiv preprint arXiv:2407.03387*, 2024.

[9] Tao Lei, Regina Barzilay, and Tommi Jaakkola. Rationalizing neural predictions. *arXiv preprint arXiv:1606.04155*, 2016.

[10] 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 Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz, editors, *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.

[11] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. Layer-wise relevance propagation: an overview. *Explainable AI: interpreting, explaining and visualizing deep learning*, pages 193–209, 2019.

[12] Sameer Pimparkhede, Mehant Kammakomati, Srikanth G. Tamilselvam, Prince Kumar, Ashok Pon Kumar, and Pushpak Bhattacharyya. DocCGen: Document-based controlled code generation. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18681–18697, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[13] Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. Synchromesh: Reliable code generation from pre-trained language models. *arXiv preprint arXiv:2201.11227*, 2022.

[14] Yiwei Qin, Kaiqiang Song, Yebowen Hu, Wenlin Yao, Sangwoo Cho, Xiaoyang Wang, Xuansheng Wu, Fei Liu, Pengfei Liu, and Dong Yu. InFoBench: Evaluating instruction following ability in large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 13025–13048, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[15] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. *CoRR*, abs/1602.04938, 2016.

[16] Suranjana Samanta and Sameep Mehta. Towards crafting text adversarial samples. *arXiv preprint arXiv:1707.02812*, 2017.

[17] Jiao Sun, Yufei Tian, Wangchunshu Zhou, Nan Xu, Qian Hu, Rahul Gupta, John Frederick Wieting, Nanyun Peng, and Xuezhe Ma. Evaluating large language models on controlled generation tasks. *arXiv preprint arXiv:2310.14542*, 2023.

[18] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.

[19] Shuyan Zhou, Uri Alon, Frank F Xu, Zhiruo Wang, Zhengbao Jiang, and Graham Neubig. Docprompting: Generating code by retrieving the docs. *arXiv preprint arXiv: 2207.05987*, 2022.