DOI: https://doi.org/10.26565/2304-6201-2025-67-10

УДК (UDC) 004.8

Omelchenko Ihor Valeriiovych
PhD student, Department of Mathematical Modeling and Data Analysis
Karazin Kharkiv National University, Svobody Sq 4, Kharkiv, Ukraine, 61022
e-mail: ihor.v.omelchenko@gmail.com;
https://orcid.org/0009-0007-4474-4916

Strukov Volodymyr Mykhailovych
PhD in Technical Sciences, Associate Professor; Head of the Department of Mathematical Modeling and Data Analysis
Karazin Kharkiv National University, Svobody Sq 4, Kharkiv, Ukraine, 61022
e-mail: volodymyr.strukov@karazin.ua;
http://orcid.org/0000-0003-4722-3159

Impact of decoding methods in LLMs on the correctness of agent action planning in virtual environments

Relevance: The knowledge and skills acquired by Large Language Models (LLMs) from training data can be applied to the task of action planning for autonomous agents. The classical approach to text generation can violate the syntax of a JSON plan, making it difficult or even impossible to parse and use such a plan. A potential solution to this problem is the application of the Grammar-Constrained Decoding (GCD) method, which restricts the set of possible texts for generation according to a specified grammar.

Goal: To investigate the impact of the Grammar-Constrained Decoding (GCD) method (with and without reasoning) compared to classical Unconstrained Decoding (UCD) on JSON schema compliance, accuracy, and planning time for various LLMs in the Minigrid virtual environments.

Research methods: Research methods are computational experiments and comparative analysis. The studied LLM sequence decoding methods are Unconstrained Decoding (UCD) and Grammar-Constrained Decoding (GCD). The planning quality metrics used were: syntactic validity (compliance with the grammar/JSON schema), planning duration, and accuracy of plan generation.

Results: This work proposes the use of Grammar-Constrained Decoding (GCD) for agent action planning tasks that utilize Large Language Models (LLMs). A dataset of plan examples was prepared for the Minigrid environments: SimpleKeyDoor, KeyInBox, and RandomBoxKey. A comparison was conducted between Unconstrained Decoding (UCD), Grammar-Constrained Decoding (GCD), and GCD with reasoning across 10 open LLMs (from the Qwen3, DeepSeek-R1, Gemma3, and Llama3.2 families). Using the GCD method ensured the validity of the generated plans according to the grammar specified by the JSON schema. A reduction in planning time was achieved for the Qwen3:4b model by a factor of 17-25 and for the Qwen3:30b model by a factor of 6-8, by limiting the number of tokens in the reasoning chains. On average, the application of the GCD decoding method improved the accuracy of plan generation.

Conclusions: This research demonstrates that the Grammar-Constrained Decoding (GCD) method is effective in action planning tasks with LLMs. The GCD method guarantees the syntactic validity of plans according to the JSON schema, which is difficult to achieve with the UCD method. The GCD method also allows for the flexible determination of the length of reasoning chains through grammar rules, thereby controlling the planning duration.

Keywords: *artificial intelligence, machine learning, deep learning, artificial neural networks, intelligent information systems, automated information systems, natural language processing, large language model, prompt, decision making, agent, virtual environment, Minigrid.*

How to quote: I. Omelchenko and V. Strukov, “Impact of decoding methods in LLMs on the correctness of agent action planning in virtual environments”, *Bulletin of V. N. Karazin Kharkiv National University, series Mathematical modelling. Information technology. Automated control systems*, vol. 67, pp. 101-112, 2025. https://doi.org/10.26565/2304-6201-2025-67-10

Як цитувати: Omelchenko I., and Strukov V. Impact of decoding methods in LLMs on the correctness of agent action planning in virtual environments. *Вісник Харківського національного університету імені В. Н. Каразіна, серія Математичне моделювання. Інформаційні технології. Автоматизовані системи управління*. 2025. вип. 67. С.101-112. . https://doi.org/10.26565/2304-6201-2025-67-10

© I. Omelchenko, V. Strukov, 2025
This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0.

1 Introduction

The use of language models in agent systems has become an active area of research [1-3]. Agents operate in environments, perceive the state of the environment through observations, and execute actions chosen from a list of valid actions for that specific environment. Upon executing a chosen action, the agent receives feedback in the form of a changed environment state and, possibly, a reward signal. In each new environment, the agent must find an optimal policy. In the case of deep reinforcement learning, the agent begins learning with limited prior information about the environment. Additional prior information about the environment can be obtained without training by using pre-trained large language models (LLMs). Language models acquire generalized world knowledge from extensive training text corpora. This knowledge can be applied to specific environments. The task of planning sequences of actions, particularly abstract ones, is of special interest [4]. The planning procedure can be performed using language models.

The use of language models as a planning module in autonomous agents requires these models to have the ability to generate sequences that strictly adhere to a given plan schema. In their early stages of development, language models emerged as free-form text generators, lacking a mechanism to constrain generation to a set of texts with a predefined structure. Input and output text is represented as a sequence of tokens. The set of available tokens is defined by a token vocabulary, which is formed by training on large text corpora such that tokens consist of the most statistically common character sequences. However, to successfully solve the planning task, language models must generate token sequences that conform to a specific grammar.

Language models are trained on the task of next-token prediction in a text sequence. Pre-trained language models can be used for various tasks without fine-tuning through the method of In-Context Learning [5], where the model receives textual demonstrations of correct behavior, based on which it determines a generalized approach for solving the task. One of the approaches to generating structured data is adding examples of structured data to the training set. This enabled language models to generate structured data such as JSON, XML, and code in many programming languages with high accuracy [6]. However, this method still allows for errors in structured data generation, which lead to parsing errors and the inability to convert the generated text into data.

The need to generate strictly structured data led to the application of grammar-constrained decoding (GCD) methods to language models [7-9]. Grammar-constrained decoding ensures that text generated by language models conforms to a predefined grammar. The GCD method uses a formal grammar to describe the valid strings in a language. To describe the formal grammar, BNF (Backus-Naur Form) is used, which is a standard notation for defining the syntax of formal languages.

GCD modifies the probability distribution of tokens from the vocabulary such that tokens forbidden by the formal grammar receive a zero probability of being chosen. The generated sequences are always valid according to the schema and plausible according to the token probability distribution computed by the language model. GCD allows one to abstract away from the implementation details of the decoding mechanism and concentrate on developing a grammar that describes the sequence's structure. The grammar is represented in a declarative form and guarantees that the generated sequences will always conform to the schema.

2 Problem formulation

2.1. General problem formulation

Let V be a finite vocabulary of tokens. A language model with parameters θ defines a conditional distribution $p_\theta(w_{i+1}|w_{1:i})$, which describes the probability of the next token $w_{i+1}\in V$ following a prefix $w_{1:i}\in V^i$, where V^i is the set of all possible prefixes of length i and $w_{1:i}=(w_1,\dots,w_i)$ is the prefix of the generated sequence of tokens.

In the case of unconstrained generation, the language model computes the probability distribution over tokens as a softmax function of the logits [10]:

$$p_\theta(w_{i+1}|w_{1:i})=\frac{exp(\ell_\theta(w_{i+1}|w_{1:i}))}{\sum_{v\in V} exp\,(\ell_\theta(v|w_{1:i}))},$$

where the logit $\ell_\theta(w_{i+1}|w_{1:i})$ is the output of the final layer of the neural network for an arbitrary token w_{i+1}. Note that the right-hand side of the given equation is the definition of the $\mathrm{softmax}(\ell)$ function for a vector of logits ℓ.

At each step i, one token is selected from the conditional distribution $p_\theta(\cdot)$. Various methods for token selection exist; the simplest is selecting the token with the maximum probability. The token selection procedure can also include a temperature parameter τ, which controls the flattening of the token probability distribution. This results in an increased probability of selecting low-probability tokens and a decreased probability for high-probability ones. A lower temperature value leads to the generation of more deterministic sequences, whereas a higher temperature results in more diverse sequences. At the temperature value of τ=0, a non-zero probability remains only for the initially most probable token, and the generation becomes deterministic.

Decoding at step i can be expressed as follows:

$$w_{i+1} \sim p_\theta^{(\tau)}(w_{i+1}|w_{1:i}) \quad \text{(stochastic sampling)}, \tag{2.1}$$

$$w_{i+1}{=}\underset{w\in V}{argmax} p_\theta^{(\tau)}(w_{i+1}|w_{1:i}) \quad \text{(deterministic sampling)}. \tag{2.2}$$

For tasks that require the generation of texts with a strict structure, such as planning tasks, a lower temperature value reduces the probability of selecting tokens that violate the structure. In the case of probabilistic token selection, a language model can be made deterministic by fixing the initialization of the random number generator. Under these conditions, for a fixed input, the language model will produce a fixed output. In such a case, the language model can be represented as a deterministic mathematical function $s_{\text{out}}{=}g_\theta(s_{\text{in}};r)$, where s_{in} and s_{out} are input and output token sequences, respectively, and r is the initialization value for the pseudorandom number generator. For a fixed r and a fixed input prefix, the model generates a deterministic output.

To solve a task in an environment, it is necessary to sequentially select and execute actions that lead to the desired goal. Actions in the environment have a sequential nature: the success of subsequent actions depends on the outcome of previous actions. The planning task can be formulated as follows. Let the agent operate in an environment with discrete time steps $t{=}0,1,2,....$ At each time step t, the language model takes the prompt s_{prompt} and the observation o_t as input and computes a new text sequence $s_{\text{out},t}$. This sequence is a tuple $(s_{\text{reasoning},t}, s_{\text{plan},t})$, where $s_{\text{reasoning},t}$ is a string containing the model's reasoning (which may be empty), and $s_{\text{plan},t}$ is a string containing an action plan formulated based on the instructions in the prompt s_{prompt} and the observation o_t.

When a language model strictly adheres to the grammar of a plan, the generated sequence takes the following form:

$$s_{\text{plan},t}{=}(a_{t,1}, a_{t,2}, \dots, a_{t,n_t}),$$

where each action $a{\in}A$, and A is the set of valid actions in the environment.

The task of plan generation imposes structural constraints on the generated output s_{out}. First, the plan must be represented as an ordered sequence of discrete actions. Second, in any specific environment, the set of valid actions may vary, and each action has its own signature — a name and a set of parameters. This imposes a requirement on the language model that the generation process must produce a sequence structured as a series of actions, where each action conforms to one of the valid action signatures.

When selecting the next token according to the probability distribution, the chosen token may violate the plan's schema. To address this problem, the Grammar-Constrained Decoding (GCD) method can be applied, which restricts the selection of tokens to only those that do not violate the grammar.

In this work, we investigate the impact of Grammar-Constrained Decoding (GCD) method on agent action planning using language models in the Minigrid virtual environment.

Let G denote a context-free grammar that specifies the valid textual sequences of plans. We denote the language as $L(G){\subseteq}V^*$, where V^* is the set of all possible strings formed from tokens in the vocabulary V. At step i, for a prefix $w_{1:i}$, we introduce the set of allowed tokens:

$$C(w_{1:i}){=}\{w{\in}V|\exists\sigma{\in}V^*: w_{1:i}\cdot w\cdot\sigma{\in}L(G)\},$$

that is, a token w is allowed if there exists some sequence continuation σ such that the concatenation of sequences $w_{1:i}\cdot w\cdot\sigma$ belongs to the language $L(G)$ with the context-free grammar G.

Then, Grammar-Constrained Decoding (GCD) restricts the possible tokens at step i to the set $C(w_{1:i})$. After applying temperature, we obtain the masked logits:

$$\ell_\theta'^{(\tau)}(w|w_{1:i}){=}\begin{cases}\ell_\theta^{(\tau)}(w|w_{1:i}), & \text{if} w{\in}C(w_{1:i}); \\ -\infty, & \text{otherwise.}\end{cases}$$

The probability distribution over tokens is obtained by applying the softmax function to the masked logits [9]:

$$p_{\theta}^{(\tau,C)}(w|w_{1:i})=\frac{exp(\ell_{\theta}'^{(\tau)}(w|w_{1:i}))}{\sum_{v\in V} exp\,(\ell_{\theta}'^{(\tau)}(v|w_{1:i}))}. \tag{2.3}$$

Decoding is performed by selecting the next token from the masked distribution (2.3) either stochastically (2.1) or deterministically (2.2).

2.2. Studied Environment

For the decision-making task, a set of environments from the Minigrid library was selected. This library provides a toolkit for creating two-dimensional environments that require sequential action planning. We used three environments of increasing complexity:

- **SimpleKeyDoor:** The agent must find and pick up a key which position is not known in advance, then find a door, navigate to it, and open it. This is a basic sequential planning task.
- **KeyInBox:** The key is located inside a box. The agent must first find and open the box, take the key, and then find and open the door. This increases the plan length.
- **RandomBoxKey:** The key can be located either inside a box or outside of it. The agent has to either find and pick up the key, or find and open the box. This creates a branching of choices.

Objects in the environment have attributes such as object type, position, and color. The observations from the environment include colors, but the actions selected by the agent do not. In the environments used, the color of the door and the key always match; therefore, color is not used in the planning schema.

2.3. Language Models and Tools

The application of language models in agents imposes several requirements. These include high speed of sequence generation to ensure agent responsiveness and the ability to execute language models on low-performance devices for use in robotic systems. The following are examples of language models that are freely available and can be executed on accelerators with low computational power. The Qwen3 family of language models [11] includes models with parameter numbers ranging from 0.6 to 235 billion. These models support a reasoning mode that allows for the dynamic scaling of computational resources to improve task performance. The Qwen3 family of language models demonstrates good performance on many benchmarks for code generation, mathematical reasoning, and agent tasks. The Gemma 3 family of language models [12] is designed to run on accessible accelerators with limited memory, such as personal computers with graphics cards. The Llama family of language models [13] is also freely available and can be executed on limited computational resources. Models from the DeepSeek R1 family [14] are trained using the distillation method from a large version of DeepSeek R1 onto models from the Qwen 2.5 and Llama 3 families. A distinctive feature of these models is that they are trained using reinforcement learning to generate long chains of reasoning.

Therefore, a set of modern open-weight language models was selected based on the following criteria: the ability to run on graphics accelerators with up to 24 GB of VRAM, the availability of quantized versions, and being instruction-tuned. The models used in the study are:

- **Qwen3:** qwen3:1.7b, qwen3:4b, qwen3:8b, qwen3:30b;
- **DeepSeek-R1:** deepseek-r1:1.5b, deepseek-r1:8b;
- **Gemma3:** gemma3:4b, gemma3:12b, gemma3n:e4b;
- **Llama3.2:** llama3.2:3b.

The Ollama software tool was used to execute the models and apply the grammar-constrained decoding (GCD) method, as it supports text generation conforming to a JSON schema.

For all language models, a modified prompt from the work [15] was used, which included a task description, a JSON schema, reasoning instructions, and examples of the correct planning.

3 Methods

3.1. Decoding Methods

The following sequence decoding methods were used in the language models. The first method was Unconstrained Decoding (UCD), where the most probable token is selected at each step. The prompt included an instruction to generate only JSON without reasoning; however, this does not guarantee the syntactic correctness of the generated text. Additional post-processing was applied to the generated text to remove the reasoning fragment, if present, and to extract the substring containing the JSON object. The second method was Grammar-Constrained Decoding (GCD) without reasoning. In this case, the model generated a string that strictly conforms to the plan's schema, which guarantees syntactic correctness and requires no additional post-processing before parsing the JSON object. The third method, Grammar-Constrained Decoding (GCD) with reasoning, involved adding an optional "reasoning" field to the JSON schema, allowing the model to generate textual reasoning before formulating the final action plan.

3.2. Plan Schema

We define a plan configuration as a structure consisting of a "reasoning" text field and a "plan" sequence of actions. A plan, $P=(a_1,\dots,a_m)$, contains from 1 to 7 actions ($1 \le m \le 7$). Each action a_i is chosen from a set of options: "explore for objects", "go to object", "pick up", "drop", "toggle". The "object" parameter is a single element, while "objects" is one or more elements from a defined set $O=\{\text{door,key,box}\}$.

3.3. Evaluation Metrics

To evaluate the correctness of a plan, we used the Mean Exact-Prefix Accuracy (MEPA) metric. This metric measures the fraction of prefixes of the generated plan that exactly match the ground-truth plan. Let the ground-truth plan be $T^{(n)}=(t_1,\dots,t_n)$, where n is the number of steps in the ground-truth plan, and the generated plan be $P^{(m)}=(p_1,\dots,p_m)$, where m is the number of steps in the generated plan. The indicator function for a correct action is:

$$c_i=\begin{cases}1, & \text{if} i \le n \wedge i \le m \wedge t_i=p_i;\\ 0, & \text{otherwise.}\end{cases}$$

Let us denote the indicator of an exact prefix match for prefix of length i as

$$\sigma_i=\prod_{j=1}^{i} c_j.$$

That is, $\sigma_i=1$ if and only if all of the first i steps match. Then, MEPA for a single example is defined as the average of these indicators over all prefixes of the ground-truth plan:

$$\text{MEPA}(T^{(n)},P^{(m)})=\frac{1}{n}\sum_{i=1}^{n}\sigma_i.$$

Note that the case $n=0$ does not occur, as our dataset does not include examples with an empty plan.

As an example of calculating the MEPA metric, if $T=(\text{explore,go to,toggle})$ and $P=(\text{explore,go to,drop})$, then $\sigma_1=1, \sigma_2=1, \sigma_3=0$. The metric value will be $(1+1+0)/3 \approx 0.67$.

For a set of K examples, we calculate the macro-averaged MEPA:

$$\text{MEPA}_{macro}=\frac{1}{K}\sum_{k=1}^{K}\text{MEPA}_k,$$

where $MEPA_k$ is the MEPA value for the k-th example. Hereafter, the metric $\text{MEPA}_{\text{macro}}$ will be denoted as MEPA.

4 Experiments

We conducted computational experiments in three environments from the Minigrid suite: SimpleKeyDoor, KeyInBox, and RandomBoxKey.

4.1. Experimental Setup

All experiments were conducted on hardware with an NVIDIA RTX 3090 GPU (24 GB VRAM). We used the Ollama 0.11.10 framework to run quantized versions of the models (Q4_K_M). To ensure reproducibility, the generation temperature parameter was set to 0, and the top_k sampling limit was set to 1. The maximum length of the generated sequence was limited to 4096 tokens.

We compared three decoding methods:

1. **UCD (Unconstrained Decoding):** The model was instructed to generate only JSON. The output was then post-processed to extract a JSON object.
2. **GCD (Grammar-Constrained Decoding):** Generation was constrained by a JSON schema for the plan, which did not include a field for reasoning.
3. **GCD+R (GCD with Reasoning):** The JSON schema included an optional "reasoning" field, allowing the model to generate reasoning before the plan.

4.2 SimpleKeyDoor Environment

For the SimpleKeyDoor environment, there are six unique abstract states of the environment. When colors are taken into account, this results in 31 examples with a correct plan. The task of the language model is to generate a plan based on an observation that most closely matches the ground-truth plan.

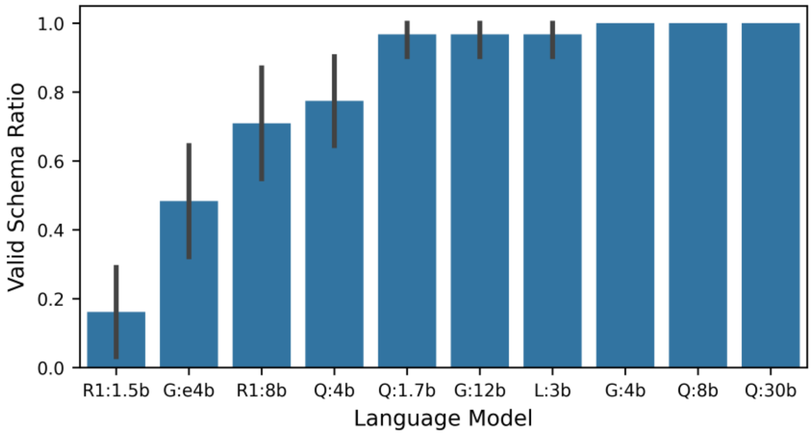

Fig. 4.1 Fraction of correctly generated plans according to the schema for different language models. The black vertical bars represent the 95% confidence interval. Legend for language models: Q:1.7b — Qwen3:1.7b, Q:4b — Qwen3:4b, Q:12b — Qwen3:12b, Q:30b — Qwen3:30b, G:4b — Gemma3:4b, G:12b — Gemma3:12b, G:e4b — Gemma3n:e4b, L:3b — Llama3.2:3b, R1:1.5b — DeepSeek-R1:1.5b, R1:8b — DeepSeek-R1:8b.

Рис. 4.1 Частка коректно згенерованих планів відповідно до схеми для різних мовних моделей. Чорні вертикальні лінії позначають 95% довірчий інтервал. Умовні позначення мовних моделей: Q:1.7b — Qwen3:1.7b, Q:4b — Qwen3:4b, Q:12b — Qwen3:12b, Q:30b — Qwen3:30b, G:4b — Gemma3:4b, G:12b — Gemma3:12b, G:e4b — Gemma3n:e4b, L:3b — Llama3.2:3b, R1:1.5b — DeepSeek-R1:1.5b, R1:8b — DeepSeek-R1:8b.

When using the UCD method (Fig. 4.1), not all models were able to generate syntactically correct JSON, which made further processing impossible. The DeepSeek-R1:1.5b and Gemma3n:e4b models proved to be the least reliable. The GCD method, by definition, guarantees 100% schema correctness.

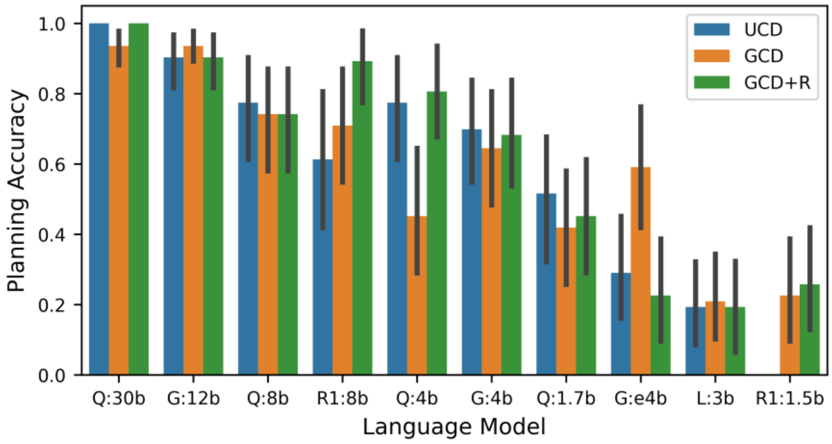

Fig. 4.2 MEPA for different language models and generation methods. The legend and confidence intervals are the same as in Fig. 4.1

Рис 4.2 МЕРА для різних мовних моделей та методів генерації. Умовні позначення та довірчі інтервали збігаються з такими для Рис. 4.1

Figure 4.2 shows that for this relatively simple task, the most powerful models (Qwen3:30b, Gemma3:12b) achieve nearly perfect accuracy regardless of the decoding method. This indicates that the task is comparatively simple for them. Meanwhile, for smaller models such as Gemma3n:e4b, the GCD method slightly improves the result. Overall, for 7 out of 10 models, the planning performance either remained unchanged or improved.

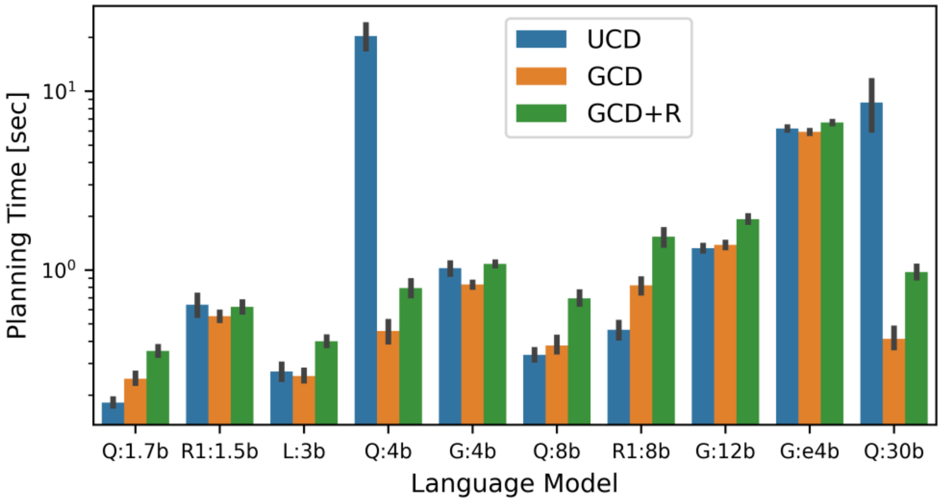

Fig. 4.3 Average generation time for a single plan for different language models and generation methods. The legend and confidence intervals are the same as in Fig. 4.1

Рис. 4.3 Середній час генерації одного плану для різних мовних моделей та методів генерації. Умовні позначення та довірчі інтервали збігаються з такими для Рис. 4.1

The vertical axis of Fig. 4.3 shows the plan generation time on a logarithmic scale. As is evident from this figure, the UCD method for the Qwen3:4b and Qwen3:30b models was significantly slower: by a factor of ≈ 25 for Qwen3:4b and ≈ 8 for Qwen3:30b. This is because these models, despite instructions not to generate reasoning, produced long chains of reasoning before the JSON response. The GCD method causes the Qwen3:4b and Qwen3:30b models to immediately generate a structured result, which significantly reduces planning time and makes these models more suitable for real-time agent systems.

4.3. KeyInBox Environment

In the KeyInBox environment, the planning complexity increases as additional steps appear: find the box, open it, and only then take the key and open the door.

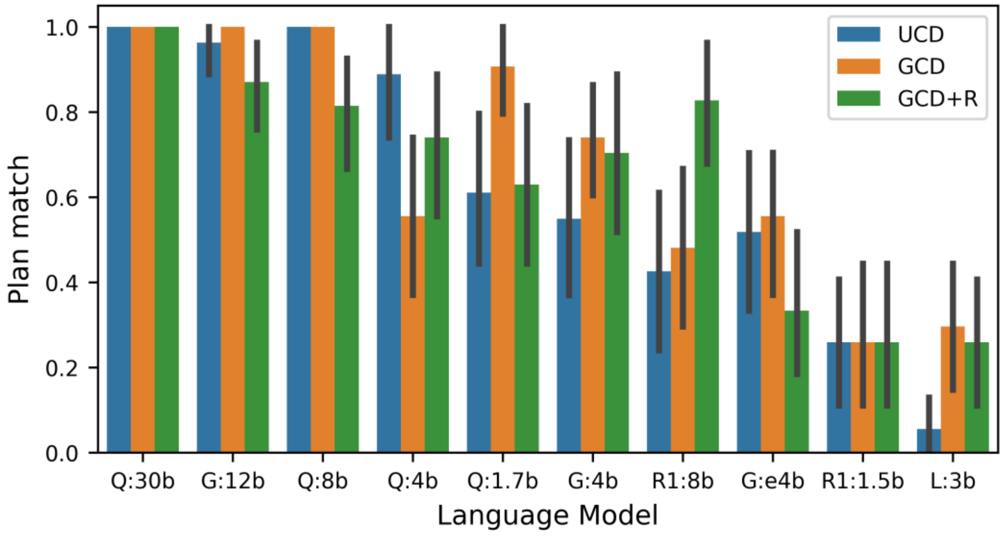

Fig. 4.4 MEPA for the KeyInBox environment for different language models and generation methods. The legend and confidence intervals are the same as in Fig. 4.1

Рис. 4.4 MEPA для середовища KeyInBox для різних мовних моделей та методів генерації. Умовні позначення та довірчі інтервали збігаються з такими для Рис. 4.1

Fig. 4.4 demonstrates that models such as Qwen3:30b continue to show high accuracy. The result for the DeepSeek-R1:8b model is particularly interesting: its accuracy significantly increases when using

GCD with reasoning. This may indicate that for models trained to generate long chains of reasoning, providing a special "reasoning" field within the JSON schema improves the quality of the final plan. Overall, for 9 out of 10 models, the planning results remained unchanged or improved.

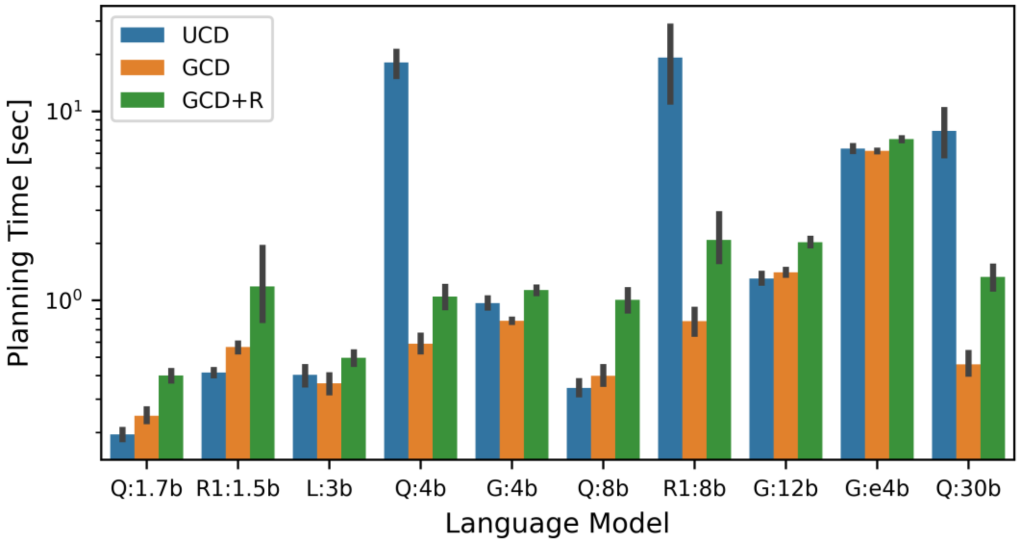

Fig. 4.5 Average single-plan generation time for the KeyInBox environment across different language models and generation methods. The legend and confidence intervals are the same as in Fig. 4.1

Рис. 4.5 Середній час генерації одного плану для середовища KeyInBox для різних мовних моделей та методів генерації. Умовні позначення та довірчі інтервали збігаються з такими для Рис. 4.1

Fig. 4.5 confirms the trend observed in the previous experiment: the Qwen3:4b and Qwen3:30b models take significantly more time to generate a plan using the UCD method due to the generation of redundant reasoning, whereas GCD ensures a fast and predictable response time. For this environment, unlike the previous one, this phenomenon is also observed for the DeepSeek-R1:8b model. Unconstrained generation was approximately 17 times slower for the Qwen3:4b model, 6 times slower for Qwen3:30b, and 9 times slower for DeepSeek-R1:8b.

4.4. RandomBoxKey Environment

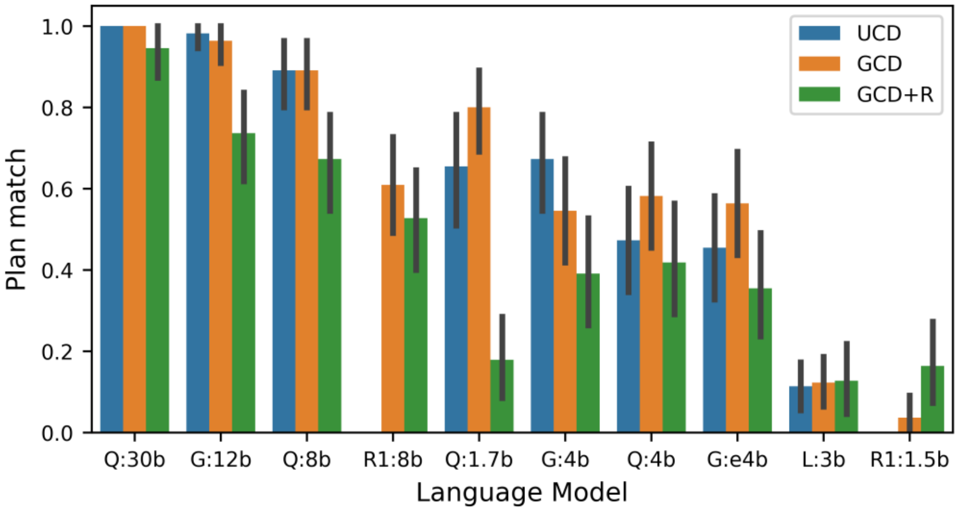

Fig. 4.6 MEPA for the RandomBoxKey environment for different language models and generation methods. The legend and confidence intervals are the same as in Fig. 4.1

Рис. 4.6 MEPA для середовища RandomBoxKey, різних мовних моделей та методів генерації. Умовні позначення та довірчі інтервали збігаються з такими для Рис. 4.1

The results in Fig. 4.6 highlight the importance of GCD in complex tasks. Both DeepSeek-R1 models failed completely with unconstrained generation, as they exceeded the 4096 token limit by generating redundant reasoning. GCD not only allowed them to generate a response but also significantly improved accuracy. For models like Qwen3:30b, accuracy remains high, but the speed advantage of GCD becomes significant. Overall, for 8 out of 10 models, the planning accuracy either did not change or improved.

Similar to the results for the previous two environments, Fig. 4.7 shows a significantly longer planning time for the Qwen3:4b and Qwen3:30b models when using the UCD method, making this approach impractical for complex tasks. Unconstrained generation was ≈ 22 times slower for the Qwen3:4b model and ≈ 6 times slower for Qwen3:30b.

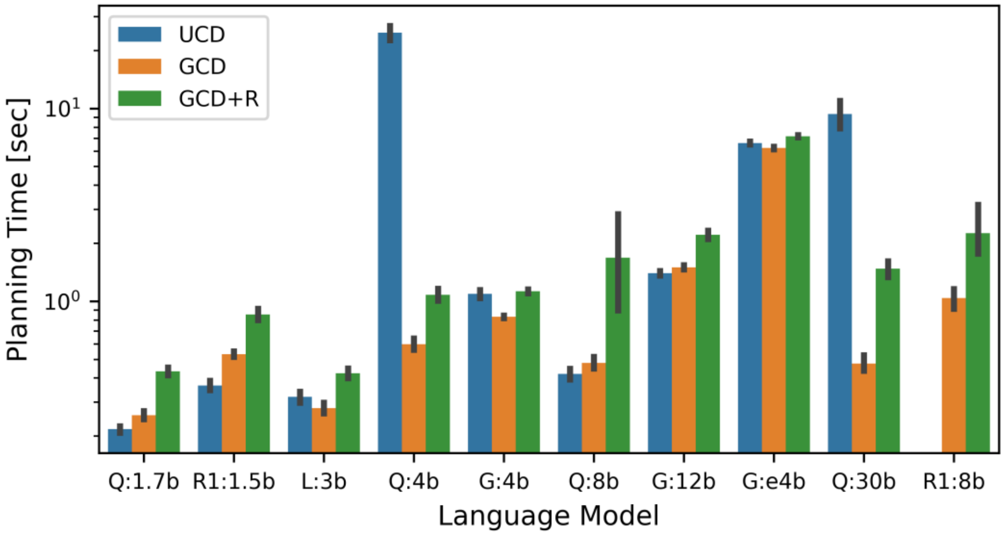

Fig. 4.7 Average generation time for a single plan for the RandomBoxKey environment for different language models and generation methods. The legend and confidence intervals are the same as in Fig. 4.1
Рис. 4.7. Середнєє значення часу генерації одного плану для середовища RandomBoxKey для різних мовних моделей та методів генерації. Умовні позначення та довірчі інтервали збігаються з такими для Рис. 4.1

Table 1. A comparison of different LLM mean MEPA precentage improvement

Табл. 1 Порівняння середнього значення відсоткового відносного покращення MEPA для різних LLM

LLM	Mean MEPA improvement, %
deepseek-r1:1.5b	-
llama3.2:3b	151.2
deepseek-r1:8b	69.9
gemma3n:e4b	44.9
qwen3:1.7b	19.4
gemma3:4b	4.5
qwen3:4b	3.5
gemma3:12b	1.8
qwen3:30b	0
qwen3:8b	-1.4

5 Discussion

Table 1 demonstrates the average relative percentage improvement for the MEPA metric across three environments for various language models using the GCD method relative to the baseline (results of the UCD method). The DeepSeek-R1:1.5b model had a zero MEPA value for the UCD baseline, making it impossible to calculate the relative improvement. For models with high MEPA value (Qwen3:8b, Qwen3:30b, Gemma3:12b), the increase is not significant, as their results were already close to the maximum.

The conducted experiments lead us to several key conclusions about the impact of the GCD method on the accuracy, schema compliance, and action planning time of agents using language models.

The UCD method, despite explicit instructions in the prompt prohibiting reasoning, cannot guarantee the absence of reasoning or control the length of reasoning chains. Many models, especially smaller ones (DeepSeek-R1:1.5b, Gemma3n:e4b), often generated syntactically incorrect JSON or added redundant reasoning, which made automatic parsing of the result impossible. This creates significant obstacles for building stable agentic systems. In contrast, the grammar-constrained decoding (GCD) method completely solves this problem by guaranteeing 100% syntactic validity of the output. This is a crucial advantage for automated data processing systems.

One of the most significant results is the substantial acceleration of the planning process when using GCD. Models prone to generating extensive reasoning (in particular, the Qwen3:4b and Qwen3:30b models for all studied environments, and DeepSeek-R1:8b for the KeyInBox environment) demonstrated significantly longer generation time with the UCD method. By constraining the output to a strict JSON schema, GCD allows for controlling the presence or length of the reasoning chain. This makes language models much more suitable for systems that require real-time decision-making.

In simple environments (SimpleKeyDoor), the advantages of GCD in planning accuracy were minor for relatively large models. However, as the task complexity increased, the impact of structured generation became more noticeable. In the most complex environment (RandomBoxKey), the unconstrained approach led to the complete failure of the DeepSeek-R1 models due to exceeding the token limit. GCD not only made it possible to obtain a response from them but also significantly increased its correctness.

6 Conclusions

This study accomplished the following. We proposed the use of Grammar-Constrained Decoding (GCD) for agent action sequence planning tasks in virtual environments, as an alternative to the classic Unconstrained Decoding (UCD) method. A dataset containing examples of correct action plans was prepared for three Minigrid environments. Computational experiments were conducted to generate agent action plans in environments from the Minigrid suite using UCD, GCD, and GCD with reasoning. The performance of these methods was evaluated based on the following metrics: syntactic validity, planning duration, and plan generation accuracy. Finally, we analyzed and compared the results of applying the UCD, GCD, and GCD with reasoning methods to the agent action planning task across the three Minigrid environments.

The results of the study demonstrated that, unlike the classic UCD method, applying the GCD method ensures that the generated sequence conforms to the specified plan grammar. This eliminates syntax errors and guarantees the successful syntactic parsing of the generated JSON plan. This outcome is particularly significant for relatively small language models, such as DeepSeek-R1:1.5b and Gemma3n:e4b. When using the classic UCD method, these models generate grammatically incorrect sequences in more than 50% of cases, whereas applying GCD guarantees adherence to the grammar.

Measurements of planning duration indicate that the classic UCD method does not guarantee a limit on the number of tokens in the generated sequence; specifically, a significant number of tokens are generated in reasoning chains. Some language models ignore instructions that prohibit the generation of long reasoning chains. Applying the GCD method resulted in a significant reduction in planning time compared to the UCD method across various environments for the following models: the planning duration for the Qwen3:4b model decreased by a factor of 17–25, for the Qwen3:30b model by a factor of 6–8, and for the DeepSeek-R1:8b model by a factor of 9 (in the KeyInBox environment). The application of GCD led to this substantial reduction in planning time by constraining the length of the reasoning chains.

Applying the GCD method improves, on average, the plan generation accuracy as measured by the MEPA metric for most models. The most significant accuracy gain was observed for the DeepSeek-R1 models. When using the UCD method, these models generated excessively long reasoning chains, which led to exceeding the token limit and resulted in failed plan generation. In contrast, with the GCD and GCD with reasoning methods, these models were guaranteed to generate a plan.

This research has shown that applying GCD improves, on average, the plan generation accuracy for most of the models tested and guarantees the syntactic validity of the generated plan according to the JSON schema. In the case of LLMs that already demonstrate high accuracy, the main benefit of GCD is the reduction in planning time rather than an improvement in accuracy. Therefore, the use of the GCD method is beneficial for enhancing the performance of language models in planning tasks.

REFERENCES

1. I. Dasgupta et al., "Collaborating with language models for embodied reasoning", arXiv [cs.LG]. 2023. [Online]. Available: https://arxiv.org/abs/2302.00763.
2. W. Huang et al., "Inner Monologue: Embodied Reasoning through Planning with Language Models", arXiv [cs.RO]. 2022. Available: https://arxiv.org/abs/2207.05608.
3. B. Hu, C. Zhao, P. Zhang, et al., "Enabling Intelligent Interactions between an Agent and an LLM: A Reinforcement Learning Approach", Reinforcement Learning Journal, Vol. 3, P. 1289–1305, 2024.
4. R. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning", Artificial Intelligence, Vol. 112, P. 181–211, 1999.
5. T. B. Brown et al., "Language Models are Few-Shot Learners", arXiv [cs.CL]. 2020. [Online]. Available: https://arxiv.org/abs/2005.14165.
6. S. Minaee et al., "Large Language Models: A Survey", arXiv [cs.CL]. 2025. [Online]. Available: https://arxiv.org/abs/2402.06196.
7. Y. Dong et al., "XGrammar: Flexible and Efficient Structured Generation Engine for Large Language Models", rXiv [cs.CL]. 2025. [Online]. Available: https://arxiv.org/abs/2411.15100.
8. S. Geng, M. Josifoski, M. Peyrard, and R. West, "Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning", arXiv [cs.CL]. 2024. [Online]. Available: https://arxiv.org/abs/2305.13971.
9. L. Beurer-Kellner, M. Fischer, and M. Vechev, "Guiding LLMs The Right Way: Fast, Non-Invasive Constrained Generation", arXiv [cs.LG]. 2024. [Online]. Available: https://arxiv.org/abs/2403.06988.
10. K. Murphy, "Probabilistic machine learning: an introduction", MIT press, 2022.
11. A. Yang et al., "Qwen3 Technical Report", arXiv [cs.CL]. 2025. [Online]. Available: https://arxiv.org/abs/2505.09388.
12. G. Team et al., "Gemma 3 Technical Report", arXiv [cs.CL]. 2025. [Online]. Available: https://arxiv.org/abs/2503.19786.
13. A. Grattafiori et al., "The Llama 3 Herd of Models", arXiv [cs.AI]. 2024. [Online]. Available: https://arxiv.org/abs/2407.21783.
14. DeepSeek-AI et al., "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning", arXiv [cs.CL]. 2025. [Online]. Available: https://arxiv.org/abs/2501.12948.
15. I. Omelchenko and V. Strukov, "On the impact of prompts on agent performance in a virtual environment", Bulletin of V. N. Karazin Kharkiv National University, series Mathematical modelling. Information technology, Automated control systems, Vol. 65, P. 56–63, 2025.

Омельченко Ігор Валерійович *Аспірант, кафедра математичного моделювання та аналізу даних*
Харківський національний університет ім. В.Н. Каразіна. майдан Свободи, 4,
Харків, Харківська область, 61022

Струков Володимир Михайлович *к.т.н., доцент; завідувач кафедри математичного моделювання та аналізу даних*
Харківський національний університет ім. В.Н. Каразіна. майдан Свободи, 4,
Харків, Харківська область, 61022

Дослідження впливу методів декодування у мовних моделях на коректність планування дій агентів у віртуальних середовищах

Актуальність. Знання та навички, отримані великими мовними моделями (LLM) з навчальних даних, можуть бути використані в задачі планування дій автономних агентів. Класичний підхід до генерації тексту може порушувати синтаксис JSON-плану, що ускладнює або робить неможливим синтаксичний розбір та використання такого плану. Можливий підхід до вирішення цієї проблеми полягає у застосуванні методу декодування з обмеженням граматики (GCD), що обмежує множину можливих текстів для генерації відповідно до заданої граматики.

Мета. Дослідити вплив методу декодування з обмеженням граматики GCD (з міркуваннями та без) порівняно з класичним необмеженим декодуванням UCD на відповідність JSON-схемі, точність та час планування дій різними LLM у віртуальних середовищах Minigrid.

Методи дослідження. Методи дослідження: обчислювальний експеримент, порівняльний аналіз. Методи декодування послідовностей в LLM: Unconstrained Decoding (UCD), Grammar-Constrained Decoding (GCD). Використані метрики якості планування: синтаксична валідність (відповідність граматиці/JSON-схемі), тривалість та точність планування.

Результати. Запропоновано використовувати метод декодування з обмеження граматики (GCD) в задачах планування дій агентів з використанням великих мовних моделей (LLM). Підготовлено датасет з прикладами планів для середовищ Minigrid: SimpleKeyDoor, KeyInBox, RandomBoxKey. Проведено порівняння методів Unconstrained Decoding (UCD), Grammar-Constrained Decoding (GCD) та GCD з міркуваннями для 10 відкритих LLM (сімейств Qwen3, DeepSeek-R1, Gemma3, Llama3.2). Використання методу GCD забезпечило валідність згенерованого плану відповідно до граматики, заданої JSON-схемою. Досягнуто скорочення часу планування для моделей Qwen3:4b у 17-25 разів, для Qwen3:30b — у 6-8 разів за рахунок обмеження кількості токенів в ланцюжках міркувань. У середньому застосування методу декодування GCD покращило точність генерації плану.

Висновки. Дослідження демонструє, що застосування методу декодування з обмеженням граматики (GCD) є доцільним в задачах планування дій з використанням LLM. Метод GCD гарантує синтаксичну валідність планів відповідно до JSON-схеми, що складно досягти з методом UCD. Метод GCD дозволяє гнучко визначати довжину ланцюжків міркувань через правила граматики і тим самим контролювати тривалість планування.

Ключові слова: *штучний інтелект, машинне навчання, глибоке навчання, штучні нейронні мережі, інтелектуальні інформаційні системи, автоматизовані інформаційні системи, обробка природної мови, велика мовна модель, промпт, прийняття рішень, агент, віртуальне середовище, Minigrid.*

Надійшла у першій редакції 09.09.2025, в останній - 10.10.2025.
The first version has been received on 09.09.2025, the final version – on 10.10.2025.