ASSESING THE STRUCTURED DATA INFERENCE CAPABILITY OF LOCAL LANGUAGE MODELS ON EDGE DEVICES FOR IOT SYSTEM COORDINATION

PHAN VAN NAM[1,*], DO HUU SON[2]

[1]*Posts and Telecommunications Institute of Technology, Ha Noi, Vietnam*
[2]*Ha Noi University of Science and Technology, Ha Noi, Vietnam*
Email address: nampv0903@gmail.com
**Corresponding author: Phan Van Nam*
http://doi.org/10.51453/3093-3706/2026/1466

ARTICLE INFO

Received: 10/02/2026
Revised: 15/4/2026
Published: 28/4/2026

KEYWORDS

Edge Computing; Local Language Model; Function Calling; JSON-formatted commands; Artificial Intelligence of Things (AIoT).

ABSTRACT

The current dependence of IoT systems on cloud computing is causing many limitations regarding latency, security risks, and stability. To overcome this, the study proposes shifting processing capabilities to edge devices using Local Language Models (Local LLMs). To resolve the conflict between the free text of LLMs and central control systems, we apply an output format coercion technique, forcing the model to interpolate intent and output data strictly adhering to the JSON command structure. Experiments on simulated high-performance edge device hardware show that this method completely eliminates format hallucination, achieving a 100% successful JSON parsing rate. Regarding performance, the token generation speed (TPS) ranges from 37 to over 184 tokens/second, the time to first token (TTFT) is under 5 ms, the end-to-end latency takes only 0,16 to 1,2 seconds, and the peak VRAM consumption is under 4,5 GB. These results excellently meet the real-time standards of smart home experiences, affirming the feasibility of the local AIoT coordination solution.

1. INTRODUCTION

1.1. The Shift from Cloud IoT to Edge AI

Over the past decade, the architecture of Internet of Things (IoT) systems has largely been built upon the Cloud Computing model. Accordingly, end devices collect data and transmit it entirely to central servers for processing, analysis, and control decision-making. Although Cloud IoT offers advantages in large-scale computing capacity and massive storage capabilities, this model is revealing serious bottlenecks. Assessing this context, a comprehensive study by Tran Thi Thuy and colleagues on smart home models over the past 10 years [1] indicated that current IoT technology platforms remain largely “limited and fragmented,” while facing numerous vulnerabilities and major challenges in both development and practical operation.

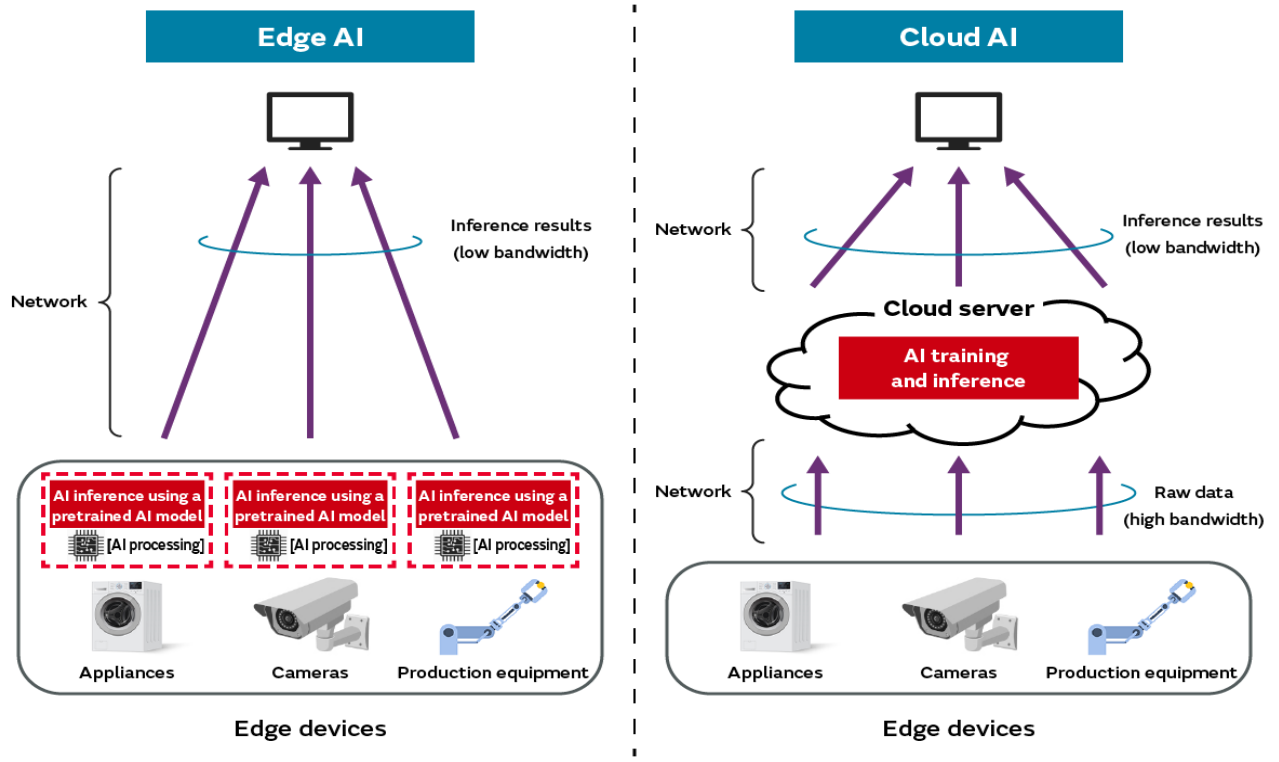

Figure 1: Comparison of Edge AI and traditional Cloud IoT architecture [2]

The first core limitation exacerbating this fragmentation is network latency. Fluctuating response times significantly degrade user experiences in smart home applications and pose risks in automated control tasks. Furthermore, total reliance on external networks causes the entire system to paralyze immediately during cable breaks or internet service disruptions.

Alarmingly, continuously transmitting raw data from sensitive sensors like security cameras or microphones to the cloud poses immense security and privacy challenges. Data leaks from smart home devices are an alarming reality [3]. Especially in Vietnam, Decree 13/2023/ND-CP establishes a strict legal corridor, forcing IoT developers to face tight constraints when collecting, processing, and transferring user data outside the local scope [4].

To resolve these bottlenecks, decentralizing computational resources to the network edge is an inevitable architectural shift [5]. Analysis reports affirm that bringing artificial intelligence to local processing at edge devices is the core solution for next-generation IoT [6]. In Vietnam, this approach is strongly applied to overcome current smart home vulnerabilities. Instead of pushing data to remote servers, the local Edge Gateway autonomously makes decisions. Shifting from Cloud IoT to Edge AI eliminates latency barriers and ensures continuous operation during internet outages, known as local execution [7]. More importantly, retaining all sensitive data within the local area network (LAN) thoroughly solves privacy and information security issues for users [8].

1.2. Technical Bottleneck Regarding the Conflict Between Natural Language and Stuctured Data

Although integrating Language Models (LLMs) at edge devices brings many architectural advantages, practical deployment encounters a core technical barrier: the direct data format conflict between artificial intelligence (AI) models and automated control systems. Inherently, LLMs operate

Email: sjttu@tqu.edu.vn

based on an autoregressive token generation mechanism [9], resulting in default outputs that are always free, unstructured text strings. Meanwhile, central IoT controllers operate based on deterministic sets of logic rules. These systems lack the capability to analyze open text semantics and can only receive and parse data blocks that adhere to a strict schema, most commonly the JSON format.

Numerous studies have proven that relying solely on prompting techniques to require the model to output structured data is highly unstable [10]. The lack of constraint mechanisms at the token generation level exacerbates the "Hallucination" phenomenon, the tendency of models to generate inaccurate, non-existent content or break predefined rules. Comprehensive surveys by Ji et al. [11], Huang et al. ([12], and Rawte et al. [13] all agree that hallucination is an intrinsic flaw of LLMs that is difficult to eliminate. In the context of JSON command generation, hallucinations often manifest as the insertion of conversational padding characters (e.g., "Here is your JSON code:"), omitted brackets, or fabricated data fields that do not exist within the system.

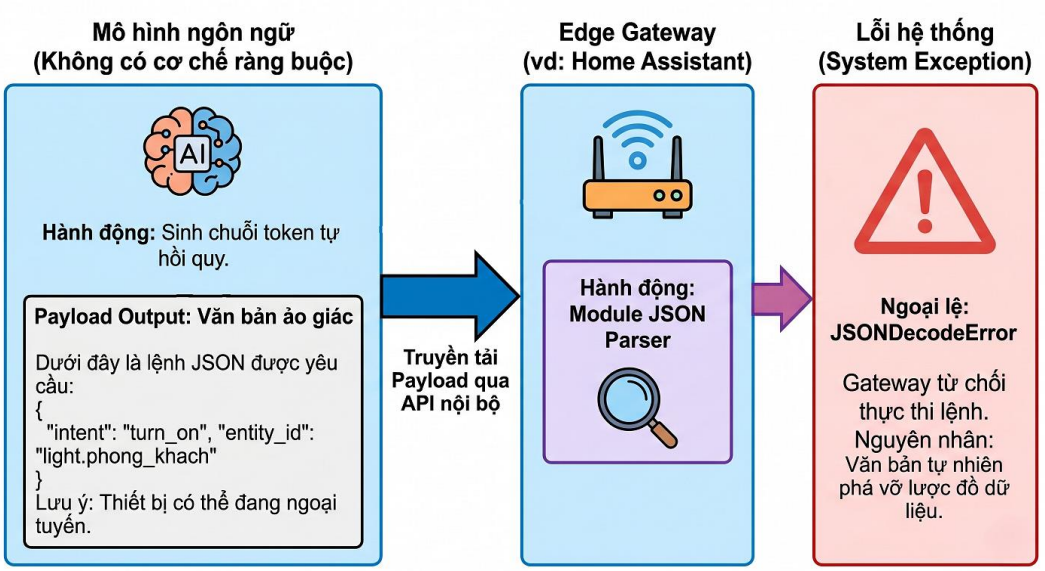

Figure 2: The bottleneck in LLM interacting with the Gateway

This challenge becomes even more severe for Small Language Models (SLMs). To be deployed locally on resource-constrained edge devices, SLMs must significantly sacrifice parameter count (typically under 8 billion parameters) [14]. This reduction leads to a severe decline in complex logical reasoning capabilities and the ability to strictly adhere to formatting compared to larger models [15], [16]. The consequences of this structural deviation have been emphasized in studies on the tool-use capabilities of LLMs. According to Schick et al. [17] and Qin et al. [18], faulty syntax generated by models will cause API systems to reject parsing, leading to a complete breakdown in the coordination chain, paralyzing the decision-making capability of the entire IoT ecosystem.

1.3. Research Objectives and Scope

Stemming from the core technical challenges and severe security barriers analyzed, this study is conducted with the overall objective of comprehensively evaluating the interpolation capacity and feasibility of deploying Local LLMs as the central coordinating brain for the IoT ecosystem. Rather than focusing on open natural language communication capabilities, the paper's core emphasis is on quantitatively measuring the effectiveness of output format coercion techniques

when the system must process diverse user control command streams. Specifically, the research evaluates based on two strict sets of criteria:

Structural Accuracy, quantified by the JSON parsing success rate. This is a decisive metric to verify the model's ability to completely eliminate hallucination phenomena, the biggest barrier causing commands to be rejected by the Gateway. Ensuring the model consistently outputs standard JSON strings, free of extra characters or fabricated data fields, is a prerequisite for maintaining seamless communication between the intent recognition system and end hardware devices.

Real-time Response Performance, analyzed deeply through detailed latency metrics, including Time To First Token (TTFT) and End-to-End Latency. In the context of smart homes or industrial control, millisecond latency directly affects practical experience. Therefore, Edge AI solutions must demonstrate immediate reflex capabilities, equivalent to or surpassing traditional Cloud solutions.

Regarding scope and limitations, the experimental design of this paper is bounded by establishing a solid baseline benchmark on a simulated high-performance edge device environment. Purposefully setting up a powerful environment at this evaluation stage is a methodological step aimed at "isolating variables". Specifically, this approach completely excludes physical latencies arising from computer resource bottlenecks. Consequently, the study can most transparently and accurately measure the intrinsic performance limits of the SLMs themselves as well as the resource consumption of the JSON grammar constraint algorithm. The ideal benchmark obtained from this experiment will serve as a complete technical proof, affirming the correctness and superiority of the IoT coordination framework using local LLMs from a software perspective.

2. RELATED WORKS

2.1. Small Language Models

Table 1: Characteristics of some typical small language models

Model Name	Parameters	Notes / Key Features
Arcee-VyLinh	~ 3B	Vietnamese model, Qwen2.5-3B architecture, context length ~32K tokens.
Vintern-1B	~ 1B	Multimodal Vietnamese model, combining image + text processing (OCR, VQA).
ViDeBERTa_base	~ 86M	Vietnamese model, DeBERTa architecture, good performance in NLU (POS tagging, NER, QA, …).
TinyLlama	~ 1.1B	Scaled-down version of Llama, optimized for Edge devices / resource-constrained machines.
Phi-3-mini	~ 3.8B	Microsoft's new small model, supports long context, good performance across many benchmarks.
Mistral 7B	~ 7B	Although "small" by some definitions, it requires larger

		resources than the ~1–3B group; high performance.

Small Language Models (SLMs) are essentially scaled-down versions of Large Language Models (LLMs), operating based on maintaining the core Transformer architecture while significantly reducing the neural network scale and parameter count. Instead of possessing tens or hundreds of billions of parameters with expensive operating costs, SLMs are designed with sizes ranging from a few million to a few billion parameters. This compactness makes SLMs particularly suitable for direct deployment on edge devices or in hardware-constrained environments.

In the current edge computing context, the segment from 1.5B to 8B parameters is becoming the ideal standard for local operation. Notable representatives include TinyLlama (~1.1B), explicitly optimized for edge devices [19], Microsoft's Phi-3-mini (3.8B) supporting long contexts while ensuring performance [20], and Mistral 7B - a model categorized as “small” but requiring slightly higher resources in exchange for superior performance [21]. Recently, advanced open-source architectures like Qwen2 [22] or Llama 3 8B [23] have also demonstrated impressive interpolation capabilities in the low-parameter segment, surpassing many previous limits of SLMs.

However, designing and applying SLMs always involves a core trade-off between model size and reasoning capacity. On the advantage side, due to the smaller matrix structure of the neural network, SLMs require significantly fewer multiplication and addition operations, resulting in extremely fast inference speeds. Simultaneously, these models use very modest RAM and consume little power, allowing them to operate stably on edge devices like Raspberry Pi, smartphones, or industrial IoT computers [24].

Despite their physical flexibility and optimization, parameter reduction entails certain limitations. SLMs cannot fully replace LLMs in tasks demanding deep logical reasoning or complex multidisciplinary knowledge synthesis. According to Zhao et al. [25], small-sized models often lack “breakthrough capabilities” that only emerge when the parameter scale exceeds tens of billions. Without proper fine-tuning or constraint techniques, SLMs can produce inaccurate results in complex scenarios. Therefore, applying SLMs as IoT coordination centers requires integrating strict formatting constraint mechanisms, such as constrained decoding for JSON output, to maximize the model's speed while masking weaknesses in semantic dispersion.

2.2. Output Format Coerion Techniques

To thoroughly understand the basis of format coercion techniques, it is first necessary to examine the core text generation principle of LLMs. Most modern LLM architectures operate on an autoregressive token generation mechanism, where the model predicts the next token based on the conditional probability of the entire preceding token sequence [9]. At each time step, the output neural layer calculates a probability distribution matrix covering the entire vocabulary space. Traditional sampling algorithms like Greedy Search, Top-k, or Nucleus Sampling (Top-p) [26] are then applied to select the subsequent token. However, these purely statistical methods lack syntactic awareness; therefore, they are highly unstable when the model is required to generate strict data structures such as source code or JSON formats [27].

To overcome this bottleneck, grammar-based sampling techniques are applied as a filter directly intervening in the token prediction phase. Instead of allowing the model to choose freely,

the system synchronizes the LLM's decoder with a Finite State Machine (FSM) or a context-free grammar rule set [28]. Specifically, the inference engine continuously matches the current state of the generating text sequence with a predefined JSON Schema. If selecting a token (e.g., a letter) violates the next syntactic rule (e.g., the system must close a bracket }), this technique directly manipulates the logit array by assigning the value of invalid tokens to -∞ [29].

This dynamic probability adjustment mechanism ensures that 100% of generated tokens strictly adhere to the target format. In real-world edge device deployments, this technique has been strongly standardized through modern inference frameworks. Typically, the llama.cpp library has successfully integrated the GBNF (Gerggl-Bakus Normal Form) grammar format to guide local models in generating accurate structures [30]. Similarly, advanced interaction frameworks like Outlines [28] or LMQL [31] also use regular expressions and FSM mechanisms to coerce output shapes. Integrating these mechanisms transforms random text generation into a predictable and fully controllable trajectory, helping SLMs overcome reasoning deficiencies to communicate safely and accurately with IoT coordination systems [32].

2.3. Coordination Structure in the IoT Ecosystem

For a Local LLM to truly act as a central coordinating “brain” replacing manual operations, it must be seamlessly integrated into the software architecture of current IoT management platforms. In today's industry-standard open-source smart home ecosystems - typically Home Assistant (HA) or OpenHAB [33], the core control system is not designed to directly receive and process natural language. Instead, the entire process of managing device states and executing services is operated through resource-oriented API communication standards based on RESTful or WebSockets protocols [34].

These API protocols require payload data packets to adhere to an extremely strict structured format according to web platform data standards [35]. A deviation of even a single character in the data field name or an incorrect data type format will cause the Gateway to throw an exception error and refuse to execute the command. To quantitatively cross-reference and evaluate the LLM's capability in this study, we define a standard JSON Schema that accurately simulates the function-calling architecture widely used to help LLMs interact with peripheral tools [18], typical of the Home Assistant ecosystem. This structure is not only the expected output format but also serves as a grammar rule set to intervene in the format coercion algorithm presented in the previous section.

The JSON schema is established with mandatory data fields to accurately map the intent from the user's Vietnamese command to physical parameters understandable by the Gateway. Specifically, it includes 3 core fields:

intent: Determines the core behavioral classification of the command. This field is typically a string with pre-listed values such as control_device or query_status. Properly classifying the intent helps the Gateway route the command to the correct processing module.

entity_id: This is the most critical information field, acting as the unique routing address of the device across the entire IoT network. The LLM must be able to semantically interpolate from the user's common naming (e.g., “the chandelier in the living room”) to a standard identifier defined by the system (e.g., light.living_room_chandelier). This field is usually constrained by regular

expressions starting with the device's domain such as light., switch., climate..

action: Specifies the particular state change or service to be applied to the entity_id. The values of action are strictly limited depending on the device type, such as turn_on, turn_off, toggle for switches/lights, or set_temperature for air conditioners.

This strict formatting structure is represented as a JSON Schema for the experimental process as follows:

```json
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "intent": {
      "type": "string",
      "enum": ["control_device", "query_status"]
    },
    "entity_id": {
      "type": "string",
      "pattern": "^(light|switch|climate|lock|fan)\\.[a-z0-9_]+$"
    },
    "action": {
      "type": "string",
      "enum": ["turn_on", "turn_off", "toggle", "set_temperature", "set_speed"]
    }
  },
  "required": ["intent", "entity_id", "action"]
}
```

Figure 3: JSON Schema structure for the experiment

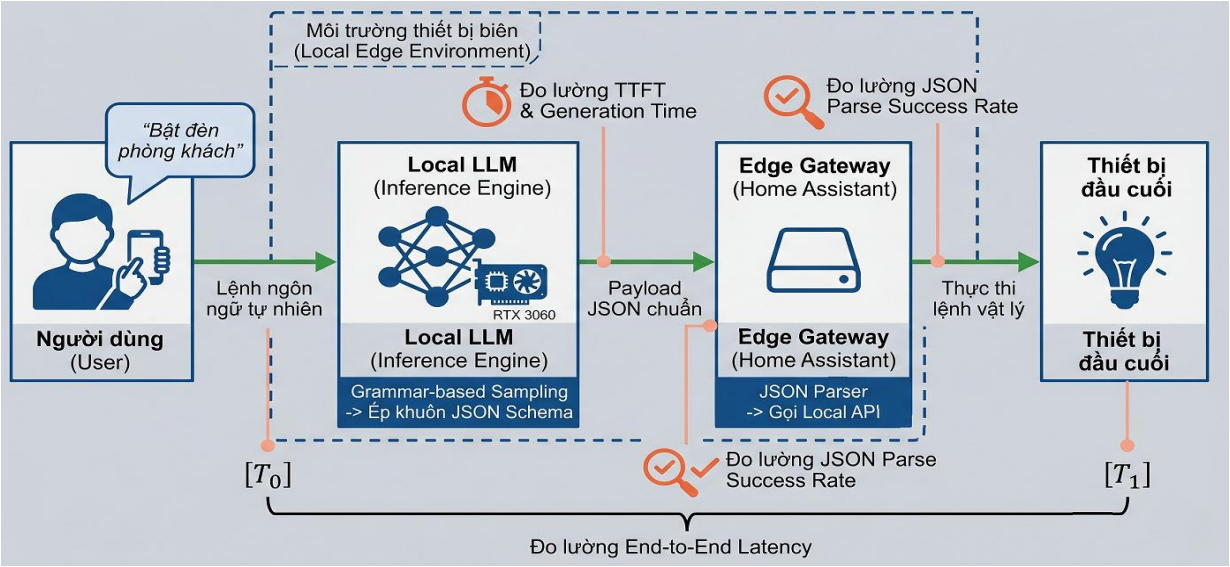

Figure 4: System's overall coordination flow diagram

3. PROPOSED METHODOLOGY

3.1. Hardware Environment

Email: sjttu@tqu.edu.vn

To accurately evaluate the interpolation capacity and measure the latency of SLMs, the experiment was conducted on a workstation system acting as a high-performance edge device. The hardware configuration was set up with an absolute focus on graphics processing capability:

Graphics Processing Unit (GPU): NVIDIA GeForce RTX 3060 (12GB VRAM). This is the core component of the entire experimental system, directly responsible for loading model weights and executing parallel matrix operations via the CUDA platform.

Host System: The system operates on an Intel Core i5-12400F processor and 16GB RAM, providing basic resources to stably maintain the Home Assistant server and JSON parsing algorithms.

The use of a workstation equipped with a discrete GPU with Tensor cores (RTX 3060) for a study on "edge devices" is an intentional methodological decision to isolate independent variables. In local AI computing, inference performance is often limited by the bulkiness of software algorithms and the poverty of hardware resources. The 12GB VRAM capacity of the RTX 3060 allows for loading the entire weights of SLMs in the 1.5B to 8B parameter segment directly into the graphics memory core, thoroughly solving the bandwidth bottleneck caused by continuous data exchange with system RAM.

By eliminating physical computational bottlenecks, the benchmark obtained from the experiment will most transparently reflect the intrinsic latency of the token generation process and the time consumption of the JSON grammar coercion algorithm itself. This ideal measurement mark serves as a "maximum performance threshold," creating a reliable foundational reference basis to compare with performance degradation when the model is transferred to lower-end microprocessor platforms in real-world deployment environments.

3.2. Evaluated Language Models

To comprehensively evaluate the impact of parameter scale on structural data interpolation capacity, the study selected a representative set of SLMs. These models were selected based on two core criteria. First, having a suitable parameter size to be fully loaded into the 12GB VRAM of the RTX 3060 graphics card in a quantized format; and second, representing architectural schools in the segment from 1 billion to under 10 billion parameters. The list of experimental subjects includes the following 4 models:

Ultra-small model: TinyLlama (~1.1B parameters). This is an extremely scaled-down and optimized version specifically for edge devices or computers with limited resources [19]. Including TinyLlama in the experiment aims to find the minimum threshold at which a language model can maintain JSON format compliance.

Vietnamese-specialized model: Arcee-VyLinh (3B parameters). Developed based on the Qwen2.5 architectural platform [36], this is a model deeply fine-tuned for Vietnamese [37] with a supported context length of up to 32K tokens. In the context of an IoT system controlled by natural Vietnamese language ("Bật đèn phòng khách"), the presence of a localized model is a prerequisite to evaluate the ability to correctly understand intent before coercing the structure.

High-performance medium model: Phi-3-mini (3.8B parameters). As a new generation of small models from Microsoft, Phi-3-mini is designed to support long contexts and has

demonstrated impressive performance on many benchmarks [20]. This model represents the most ideal balance between logical reasoning capacity and hardware resource costs.

Near-large model: Mistral 7B. Although still classified in the "small" group by some definitions, Mistral 7B [21] requires a significantly larger resource threshold compared to the 1-3B group. Configuring Mistral 7B (Q4 quantized version) on the RTX 3060 acts as a "ceiling reference mark". The purpose is to verify whether the superiority in parameter count brings any significant difference in successful parsing rates compared to smaller models.

All 4 of these models will be put into the same inference pipeline (llama.cpp), subjected to the same input prompt set, and coerced by the same JSON Schema to ensure absolute fairness in measuring latency and accuracy metrics.

3.3. Software Configuration and model Tuning

To ensure consistency, reproducibility, and hardware optimization for the data structure generation task, the entire experiment was deployed on the open-source platform llama.cpp [30]. This platform was chosen due to its deep integration support for grammar-based sampling mechanisms, allowing the direct loading of JSON Schema to completely coerce the output format for the Gateway system.

For the requirement of generating machine coordination data, determinism and absolute accuracy are prerequisites. Therefore, the Temperature parameter was set to 0 to completely switch to a Greedy Decoding strategy, while simultaneously disabling Top-P and Top-K parameters. Eliminating "creativity" and randomness helps mitigate the risk of format deviation, ensuring the model always returns a consistent JSON object.

Furthermore, the context length limit was locked at 2048 tokens to prevent KV Cache bloating, helping to thoroughly save VRAM on the RTX 3060 (12GB) graphics card. This is an ideal space, just enough to accommodate system instructions, the entire JSON Schema rule set, and the Vietnamese user prompt, while maintaining token generation speed at a maximum without causing hardware bottlenecks.

3.4. Test Dataset

To evaluate the interpolation capacity of language models in the Edge IoT environment realistically, the test dataset was not randomly generated but purposefully designed based on the natural language characteristics of Vietnamese people. Based on survey studies on voice control command usage habits in the smart home ecosystem in Vietnam [38], users tend to shift from using "mechanical commands" (clear subject-verb-adverb) to "communicative commands" with high contextual and implicit meaning.

To accurately quantify the model's capabilities in both aspects: Natural Language Understanding (NLU) and JSON Format Compliance, the dataset consisting of 90 Vietnamese commands was divided into 3 scenario groups with increasing complexity:

- **Group 1: Direct commands - Difficulty: Easy**
    + **Characteristics:** Commands contain explicit information, specifically identifying

the device name (entity) and specific physical action. This group is used to evaluate the model's basic capacity in syntax parsing.

+ **Example:** “Bật đèn phòng khách.” (Turn on the living room light).

+ **Expected JSON result:** {“intent”: “control_device”, “entity_id”: “light.phong_khach”, “action”: “turn_on”}.

- Group 2: Contextual implied commands - Difficulty: Hard

+ **Characteristics:** Commands entirely lack direct action keywords (turn on/off) or device types. Users only express a feeling or describe an environmental state. This stage requires the language model to have logical reasoning capacity to interpolate from a “problem” to a “physical solution”. This will be a maximal test for Vietnamese-specialized models like Arcee-VyLinh compared to international models.

+ **Example:** “Phòng này tối quá, tôi không thấy đường.” (This room is too dark, I can't see the way). (Intent: Needs light).

+ **Expected JSON result:** {“intent”: “control_device”, “entity_id”: “light.phong_khach”, “action”: “turn_on”}.

- Group 3: Sequential/Multi-intent commands - Difficulty: Extremely Hard

+ **Characteristics:** Commands contain multiple control intents simultaneously directed at different device groups within a time sequence or context. This scenario tests the model's ability to automatically inductively group commands into group entities or its degree of “structure breaking resistance” when forced to output multiple actions within a single JSON frame.

+ **Example:** “Tôi đi ngủ đây, tắt hết đèn và khóa cửa chính lại.” (I'm going to sleep, turn off all the lights and lock the main door).

+ **Expected JSON result:** Requires the system to map to group IDs (e.g., {“intent”: “control_device”, “entity_id”: “group.all_lights”, “action”: “turn_off”}) or generate an array of sequential JSON objects if the Schema allows.

Breaking down the dataset into these 3 scenario groups helps the experiment not only answer the question “Which model outputs JSON with the fewest errors?”, but also clarifies “Which model is truly the smartest in understanding Vietnamese living habits?”. The dataset contains 90 Vietnamese commands in total, with 30 commands allocated to each group.

3.5. Measurement Metrics

To quantitatively evaluate the performance and reliability of SLMs when acting as a coordination center on edge devices, the study establishes a strict set of measurement metrics. These metrics are designed to cover both core aspects of an AIoT system: real-time response speed and data structure accuracy.

Time To First Token (TTFT): This metric measures the latency from the time the edge system finishes processing the user's input command until the model generates the first result token. In the user experience of a smart home system, TTFT represents the “cognitive waiting time”,

directly determining the perceived responsiveness of the device.

Measured in milliseconds via the internal timer of the inference framework. The calculation formula is expressed as:

$$TTFT = T_{first\_token} - T_{promp\_received} \quad (1)$$

Where T_{first_token} is the timestamp when the system records the first output token, and $T_{promp\_received}$ is the timestamp when the entire input context (including System Prompt, JSON Schema, and User Prompt) has been fully loaded into the GPU memory.

Tokens Per Second (TPS): TPS is a measure of the model's throughput, reflecting the speed of text string generation after the first token has appeared. For API command generation tasks, a high TPS ensures that the entire structured payload data block (JSON) will be completed and packaged quickly to be sent to the Gateway for physical command execution.

Calculated as the ratio between the total number of tokens generated in the output sequence and the generation time in seconds (excluding the TTFT waiting time).

$$TPS = \frac{N_{tokens}}{T_{end} - T_{first\_token}} \quad (2)$$

Where N_{tokens} is the total number of output tokens, and T_{end} is the timestamp of completing the entire result sequence.

End-to-End Latency (E2E): This is the most practical metric directly reflecting the end-user experience. E2E Latency covers the entire lifecycle of a processing flow, starting from when the edge device receives the text command, through the LLM interpolation and JSON generation process, until the Gateway successfully parses the syntax and is ready to trigger the physical API call.

Calculated as the sum of the model's text generation latency plus the parsing latency of the Gateway system.

$$E2E = T_{api\_ready} - T_{promp\_received} \quad (3)$$

Where T_{api_ready} is the timestamp when the Gateway confirms the JSON file is completely valid and places it into the command execution queue.

JSON Parse Error Rate: This is the most important metric to evaluate the reliability of applying SLMs combined with grammar sampling algorithms. A result is counted as an “Error” when the output text string falls into one of two cases: (1) Cannot be decoded into a valid JSON object (causing system exceptions like JSONDecodeError), or (2) Successfully decoded but violates data constraints (e.g., missing mandatory fields, invalid entity_id format) defined in the standard Schema previously.

Calculated as the percentage (%) of the number of rejected results by the Gateway over the total number of test commands.

$$Error_{Rate} = \left(\frac{N_{error}}{N_{total}}\right) \times 100 \quad (4)$$

Where N_{error} is the number of structurally violating output strings, and N_{total} is the total number of test scenarios in the dataset. The closer this metric is to 0, the more the model proves its safety and determinism for real-world deployment.

4. RESEARCH RESULTS

4.1. Token Generation Speed and End-to-End Latency

The experiment measuring Time to First Token (TTFT) and End-to-End Latency across 90 test scenarios (RTX 3060 12GB configuration, Q4 quantization) yielded the following average results:

Table 2: Evaluation of inference performance and End-to-End Latency (E2E Latency) by scenario group

Evaluation Model	Metric	Group 1	Group 2	Group 3
TinyLlama (~1.1B)	TPS	180,621/s	183,493/s	184,504/s
	E2E Latency	170ms	164ms	2,443ms
Arcee-VyLinh (3B)	TPS	37,891/s	37,234/s	52,044/s
	E2E Latency	662ms	687ms	1,198ms
Phi-3-mini (3.8B)	TPS	79,921/s	80,338/s	68,607/s
	E2E Latency	367ms	365ms	799ms
Mistral (7B)	TPS	57,399/s	57,650/s	57,406/s
	E2E Latency	514ms	501ms	1,233ms

The experiment on the Edge Gateway platform showed that all four models excellently met real-time standards. Thanks to cache optimization, the Time to First Token (TTFT) across all scenarios reached levels under 5ms. This confirms that the hardware processed context loading smoothly, completely eliminating user cognitive delay during the command reception phase.

End-to-End Latency increased proportionally with parameter scale, evidenced by Mistral (7B) responding three times slower than TinyLlama (1.1B) despite both using the Q4 quantization standard. However, the token generation throughput (TPS) heavily depended on neural network architecture optimization. Typically, Phi-3-mini (3.8B) achieved a throughput of approximately 80 tokens/second, helping the system respond nearly twice as fast as Arcee-VyLinh (3B) despite possessing a larger number of parameters.

The disparity between scenario groups proves that the core cause of performance degradation did not lie in the input semantic complexity but in the autoregressive bottleneck of the output sequence,. When shifting to process sequential commands (Group 3), the execution time of all models surged by 1,7 to 2,4 times compared to single commands simply because the generated JSON structure had to be longer. This affirms that in the AIoT environment, Token Per Second (TPS) is the decisive factor to maintain system continuity.

4.2. JSON Structural Integrity

By integrating the grammar-based sampling algorithm with the GBNF format, the system established a strict syntax control filter right at the model's logit output layer,. Experimental results demonstrate a breakthrough improvement in successful syntax parsing rates compared to free text generation mechanisms.

Table 3: Syntax parsing rate and semantic accuracy by scenario group

Evaluated Model	Metric	Group 1	Group 2	Group 3
TinyLlama (~1.1B)	Valid JSON syntax	100%	100%	33,33%
	Correct standard format	100%	100%	33,33%
	Correct user intention	3,33%	3,33%	0%
Arcee-VyLinh (3B)	Valid JSON syntax	100%	100%	100%
	Correct standard format	100%	100%	100%
	Correct user intention	86,67%	46,67%	33,33%
Phi-3-mini (3.8B)	Valid JSON syntax	100%	100%	100%
	Correct standard format	100%	100%	100%
	Correct user intention	30%	23,33%	0%
Mistral (7B)	Valid JSON syntax	100%	100%	100%
	Correct standard format	100%	100%	100%
	Correct user intention	53,33%	16,67%	20%

Based on the data from Table 3, integrating the grammar sampling algorithm (GBNF) brought a breakthrough change in data integrity. However, the results also clearly revealed the boundary between “generating correctly formatted code” and “understanding the correct intent”,.

Except for the ultra-small TinyLlama model, which suffered structural breaks in sequential

commands (reaching only 33.33%), the remaining three models (Arcee-VyLinh, Phi-3-mini, Mistral) achieved a 100% successful syntax parsing rate across all scenarios. System exception errors stemming from redundant/missing brackets, missing commas, or the model arbitrarily inserting natural conversational strings (such as "Yes", "Okay", "I have turned on the device") were completely eliminated thanks to the constrained decoding technique.

Although GBNF ensured the output data was a 100% accurate JSON object, the system-level successful execution rate showed deep divergence due to the interpolation capability of each LLM core. Arcee-VyLinh led with a correct intent understanding rate of 86.67% (Group 1) but plummeted to 33.33% (Group 3). Meanwhile, international models like Phi-3-mini even dropped to 0% in complex scenarios.

Most of these failures did not stem from JSON schema violations, but belonged to semantic deviation exceptions during the mapping process,. Typical technical error cases recorded include:

Identifier Hallucination: The model generated an entity_id field that fully complied with the schema's regular expression rules, but that entity did not exist in the Gateway's internal database. This error often appeared when processing highly generalized commands in Group 2 and Group 3.

Figure 5: The model generates an entity_id field that does not exist in the internal database

Action and Context Misalignment: The model correctly identified the device but wrongly interpolated the action due to a lack of understanding of the local context,. This error was most evident in Group 2 (Contextual implied commands).

Figure 6: The model incorrectly interpolates the action due to lacking context understanding

When receiving the input command "The room is too cold" (Phòng lạnh quá), the system

expects the model to call the "turn_off" action (turn off the AC) or "set_temperature" (increase the temperature). The model interpolating the keyword "cold" into the "turn_on" action would cause the system to respond in severe contradiction to the user's actual needs.

Action Omission and Incorrect Entity Merging: This is the main reason why the semantic accuracy of Group 3 (Sequential commands) dropped severely (fluctuating from 0% up to 33.33%),. When facing multi-intent commands, for example: "I'm going to sleep, turn off all the lights and lock the main door," the models (especially the ultra-small group under 2B) often faced limitations in maintaining their attention window. Consequently, the model output an early closed JSON structure right after processing the first clause (only outputting the turn_off command for the lights), completely omitting the "lock door" task. In other cases, the model illogically tried to merge two devices into a single non-existent ID (e.g., "entity_id": "light_door.bedroom"),. This proves: the GBNF filter can prevent JSON brackets from syntax errors, but is completely powerless against the breakdown of the reasoning flow in small neural networks when having to track long contexts.

Figure 7: Groundtruth (left) and TinyLlama model output (right) only outputting the first clause

By dissecting these error cases, the experiment reaffirms: The constrained decoding format coercion technique is a "necessary condition" to protect the integrity of the software platform and avoid Gateway system crashes,. However, the native natural language understanding capability of the LLM core is the "sufficient condition" to ensure the physical coordination system operates flawlessly and accurately meets user needs.

4.3. System Resource Consumption

Deploying language models on edge devices requires strict control over hardware resources, especially graphics memory (VRAM). The experiment measured VRAM consumption at two core stages: Input context processing and JSON generation, while simultaneously evaluating a continuous load test on the RTX 3060 GPU (12GB).

Table 4: Peak VRAM consumption by stage (Q4 Format, Context = 2048 tokens)

Model	Base VRAM	Context Loading VRAM	Generation VRAM	RTX 3060 Occupation Rate

Email: sjttu@tqu.edu.vn

TinyLlama (~1.1B)	874mb	52mb	926mb	7.54%
Arcee-VyLinh (3B)	2318mb	46mb	2364mb	19.24%
Phi-3-mini (3.8B)	3264mb	56mb	3320mb	27.02%
Mistral (7B)	4382mb	56mb	4438mb	36.12%

Deploying models on edge devices requires strict hardware resource control,. The experiment showed that all quantized models operated safely on the RTX 3060, with the highest peak VRAM occupation reaching only 36.12% (4438 MB) for Mistral 7B. The strategy of limiting the Context Length to 2048 tokens maximized its effectiveness by keeping the KV Cache capacity extremely low (only 46 to 56 MB), completely eliminating the risk of memory bloating when cross-referencing the JSON Schema. Notably, the system passed the load test by continuously and repeatedly processing the test dataset without memory leaks or thermal throttling, maintaining absolute stability in End-to-End latency. Consuming less than 50% of the workstation's capacity confirms the feasibility of directly integrating 1B-3B segment models (requiring under 2.5GB of VRAM) into consumer edge devices with 2-4GB of RAM, such as the NVIDIA Jetson Nano or future SBC boards integrated with NPUs.

5. CONCLUSION AND FUTURE DEVELOPMENT

This research has affirmed the feasibility and practical performance of Small Language Models (SLMs) acting as the central AIoT coordinating hub at edge devices. The core contribution of this paper is demonstrating that the constrained decoding mechanism (specifically using grammar-based sampling like GBNF) thoroughly resolves the format hallucination problem. By ensuring 100% of the output data strictly adheres to the predefined JSON structure, this technique completely eliminates exception errors that cause Gateway software platforms to crash.

Applying GBNF introduces a necessary trade-off. While it guarantees absolute data integrity, the strict cross-validation process creates an intrinsic computational latency, slightly reducing the Token Per Second (TPS) throughput and increasing the hardware load. However, in the context of AIoT edge operations, sacrificing a small fraction of hardware performance for absolute system-level stability and safety is a highly practical and mandatory trade-off.

In terms of practical feasibility, the end-to-end latency for standard control tasks (Groups 1 and 2) fluctuated between 164 ms and 687 ms, deeply within the safe zone of real-time standards and excellently surpassing the “golden standard” of under 1 second (< 1000 ms) for smart home user experiences. This confirms that integrating SLMs with GBNF formatting on the Edge Gateway is not merely a proof of concept but is completely ready for real-world consumer AIoT deployments, outperforming traditional Cloud AI solutions in both speed and privacy.

Despite the excellent structural integrity, semantic reasoning in multi-intent scenarios remains a challenge for models under 8B parameters. Therefore, future research will focus on several core optimization and development directions:

Hardware and Algorithm Optimization: Applying dynamic and deep quantization techniques to further optimize VRAM resources. This will create the premise to transfer this inference architecture to lower-end edge devices, such as pure CPU embedded systems or ARM-based integrated NPU microcontrollers (e.g., Rockchip, Snapdragon).

Model Fine-tuning: Exploring Parameter-Efficient Fine-Tuning (PEFT/LoRA) using standardized datasets of "Natural Command - JSON Structure" pairs. Fine-tuning will help SLMs implicitly memorize the structural format, reducing the processing pressure on the GBNF filter and significantly enhancing semantic accuracy in deciphering complex, multi-intent command sequences.

Multimodal Vision Integration: Upgrading the system to incorporate Multimodal Vision capabilities. This will allow the Gateway to directly analyze visual streams from the physical environment to automatically trigger proactive control scripts, removing the complete dependency on user-initiated textual or voice commands.

REFERENCES

1. Tran-Thi Thuy, Pham-Dinh Quoc, Dang-Thanh Khang, "Research on IoT applications in building smart home models in the past 10 years", *Can Tho University Journal of Science,* 2021.
2. Articles, muRata, "Understanding Edge AI: Artificial Intelligence Meets IoT," 2024. [Online]. Available: http://article.murata.com/en-global/article/what-is-edge-ai.
3. VDTS, VD Technology Solution, "Security challenges for smart devices," 2024. [Online]. Available: https://vdts.vn/thach-thuc-bao-mat-doi-voi-thiet-bi-thong-minh-security-challenges-with-smart-home-iot-devices/.
4. Law Library, Apr 17, 2023. [Online]. Available: https://thuvienphapluat.vn/van-ban/Cong-nghe-thong-tin/Nghi-dinh-13-2023-ND-CP-bao-ve-du-lieu-ca-nhan-465185.aspx.
5. Weisong Shi, Fellow, Jie Cao, Quan Zhang, Youhuizi Li, Lanyu, "Edge Computing: Vision and Challenges," *IEEE Internet Of Things Journal,* 2016.
6. Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, Junshan Zhang, "Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing," *IEEE,* pp. 1738-1762, 2019.
7. Satyanarayanan, Mahadev, "The Emergence of Edge Computing," *IEEE Computer,* pp. 30-39, 2017.
8. Pham-Ngoc Minh, Ha-Thi Hong Van, "Cybersecurity and challenges for the Internet of Things,," *Technology, Products and Life,* 2024.
9. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, "Attention Is All You Need," *Advances in neural information processing systems,* 2017.
10. Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child et al, "Language Models are Few-Shot Learners," *Advances in Neural Information Processing Systems,* 2020.
11. Ziwei Ji, Nayeon Lee, Rita Frieske, Tiehzheng Yu et al, "Survey of Hallucination in Natural Language Generation," *Computation and Language,* 2022.
12. Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang,

Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, Ting Liu, "A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions," *Arxiv: Computation and Language,* 2024.

13. Vipula Rawte, Amit Sheth, Amitava Das, "A survey of Hallucination in Large Foundation Models," *Arxiv: Artificial Intelligence,* 2023.
14. Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, Guillaume Lample, "LLaMA: Open and Efficient Foundation Language Models," *Arxiv: Computation and language,* 2023.
15. Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck et al, "Textbooks Are All You Need," *Arxiv: Computation and Language,* 2023.
16. Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, Thomas Scialom, "Augmented Language Models: a Survey," *Arxiv: Computation and Language,* 2023.
17. Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, Thomas Scialom, "Toolformer: Language Models Can Teach Themselves to Use Tools," *Arxiv: Computation and Language,* 2023.
18. Yujia Qin, Shi Liang, Yining Ye, Kunlun Zhu, Lan Yan, Ya-Ting Lu, Yankai Lin, X. Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Marc H. Gerstein, Dahai Li, Zhiyuan Liu, Maosong Sun, "ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs," *International Conference on Learning Representations,* 2023.
19. Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, Wei Lu, "TinyLlama: An Open-Source Small Language Model," *Arxiv: Computation and Language,* 2024.
20. Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Qin Cai, Vishrav Chaudhary et al, "Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone," *Arxiv: Computation and Language,* 2024.
21. Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock et al, "Mistral 7B," *Arxiv: Computation and Language,* 2023.
22. Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren et al, "Qwen Technical Report," *Arxiv: Computation and Language,* 2023.
23. AI@Meta, "Llama 3 model card," *Meta AI,* 2024.
24. Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, "AWQ: ACTIVATION-AWARE WEIGHT QUANTIZATION FOR," *Arxiv,* 2023.
25. Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu et al, "A Survey of Large Language Models," *Arxiv: Computation and Language,* 2026.
26. Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, Yejin Choi, "The Curious Case of Neural

Text Degeneration," *ICLR,* 2020.

27. Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, Sumit Gulwani, "Synchromesh: Reliable code generation from pre-trained language models," *Arxiv: Computer Science,* 2022.
28. Brandon T. Willard, Rémi Louf, "Efficient Guided Generation for Large Language Models," *Arxiv: Computation and Language,* 2023.
29. Kanghee Park, Timothy Zhou, Loris D'Antoni, "Flexible and Efficient Grammar-Constrained Decoding," *Arxiv: Computation and Language,* 2025.
30. Gerganov, "llama.cpp: Port of Facebook's LLaMA model in C/C++," GitHub repository, 2023.
31. Luca Beurer-Kellner, Marc Fischer, Martin Vechev, "Prompting Is Programming: A Query Language for Large Language Models," *PLDI'23: 44th ACM SIGPLAN International Conference on Programming Language Design and Implementation,* 2022.
32. Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A. Saurous, Yoon Kim, "Grammar Prompting for Domain-Specific Language Generation with Large Language Models," *Arxiv: Computation and Language,* 2023.
33. Carles Gomez, Stefano Chessa, Anthony Fleury, George Roussos, Davy Preuveneers, "Internet of Things for enabling smart environments: A," *HAL Open Science,* 2019.
34. Dominique Guinard, Vlad Trifa, Friedemann Mattern & Erik Wilde, "From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices," *Springer Nature,* 2011.
35. Felipe Pezoa, Juan L. Reutter, Fernando Suarez, Martín Ugarte, Domagoj Vrgoč, "Foundations of JSON Schema," *Proceedings of the 25th International Conference on World Wide Web,* pp. 263-273, 2016.
36. An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang et al, "Qwen2 Technical Report," *Arxiv: Computation and Language,* 2024.
37. Nguyen, Quan, "Introducing Arcee-VyLinh - A Powerful 3B Parameter Vietnamese Language SLM," 2024. [Online]. Available: https://www.arcee.ai/blog/introducing-arcee-vylinh-a-powerful-3b-parameter-vietnamese-language-model.
38. Bac Ninh Newspaper, "Published the first research report in the field "smarthome" in Viet Nam," 2022. [Online].