A HARDWARE-VALIDATED SYNTHETIC DATASET FOR RELIABLE IOT CODE GENERATION WITH LLMS

Simeon Monov, Mariana Veselinova

Abstract. *General-purpose Large Language Models (LLMs) often produce unreliable code for specialized IoT ecosystems. While these platforms may use familiar languages like JavaScript, they run a restricted version with specific APIs and constraints. We find that general models, heavily biased by their training on vast, traditional JavaScript codebases, frequently generate functionally incorrect scripts that fail on-device. This problem is compounded by a scarcity of domain-specific documentation and validated code examples. To address this, we present a systematic methodology for creating a specialized dataset. Starting from a seed corpus of 80 validated scripts, we used an LLM to synthetically generate a larger, diverse dataset. Crucially, every synthetic script was then subjected to rigorous, execution-based validation on target hardware to ensure functional correctness. This pipeline yielded a validated dataset of over 1,100 scripts, providing the robust foundation needed to fine-tune a model that overcomes traditional JavaScript bias and generates reliable, domain-specific code.*

Key words: Large Language Models, LLMs, IoT Automation, Specialized Dataset, LLM Training, Fine-Tuning, Code Generation, Domain-Specific, IoT, Synthetic Data, Execution-Based Validation.

1. Introduction

The rapid advancement of Large Language Models has ushered in a new era of automated code generation, providing increased code productivity across numerous domains [1]. However, this progress is facing a significant obstacle in highly specialized, resource-constrained environments in the field of Internet of Things (IoT). Shelly Group, a leading Bulgarian IoT manufacturer with global presence in over 130 countries, produces smart home automation devices that operate on modified JavaScript (Espruino) with device-specific APIs. Generating functional scripts for proprietary hardware, such as the Shelly ecosystem, demands niche-domain knowledge of specific device APIs, internal hardware constraints and networks protocols. General-purpose LLMs frequently lack this essential contextual understanding, leading to outputs characterized by code

hallucinations (e.g., calling non-existent methods or using incorrect parameters) [2, 3]. This reliability gap is a significant barrier, particularly for the vast majority of IoT device owners who lack sufficient enough knowledge in programming.

2. Challenges in Collecting Initial Data

The main impediment to training a specialized Large Language Model for IoT automation was data scarcity. The project was initiated with a highly constrained corpus of 80 scripts which were exclusively derived from authorized educational materials. While these foundational sources were functional and relevant, the volume was demonstrably insufficient for robust LLM fine-tuning. Apart from volume constraints, additional factors necessitated the need for data synthesis.

The amount of publicly available and functional scripts specifically tailored for Shelly devices is extremely limited [4]. This scarcity is due to the fact that the devices operate on a modified and more restricted version of JavaScript Espruino, requiring specialized syntax and API calls. Acquiring a dataset for sufficient LLM fine-tuning requires thousands of examples. Manual script creation by experts to produce a dataset of such magnitude is both time-consuming and financially restraining. Additionally, while official documentation exists and provides structured information, it inherently does not cover a broader spectrum of real-world usage scenarios, more complex component combinations and edge cases. Training LLMs to handle sophisticated automation tasks demands exposure to these important scenarios.

The Shelly product line's diversity introduces another technical complexity in data collection. Each Shelly device, such as the Shelly RGBWW PM (RGB and Light components), the Shelly Dimmer (dimmable light support), the Shelly 2PM (two Switch and a Cover component), possess unique capabilities. A generalized script could often be non-transferable to such devices. To ensure the LLM achieves high diversity and can adapt across a wider range of devices, the training corpus needs to explicitly incorporate these device-specific distinctions [5].

These challenges require a systematic data generation approach that can concurrently scale the dataset, guarantee functional accuracy and address domain-specific diversity, laying the foundation for a specialized training procedure outlined in the following sections.

3. Methodology for Synthetic Data Generation

To address the fundamental challenge of insufficient high-quality, domain-specific training data for the Shelly ecosystem, a systematic synthetic data generation pipeline has been developed. This methodology enabled the development of a training set of 1100 validated examples – an amount necessary for achieving robust performance in specialized code generation tasks, based on the initial corpus of only 80 verified scripts.

The data generation pipeline is designed to maintain the functional correctness and specific coding style of the target domain, consisting of four sequential stages:

1. Seed Corpus Preparation: The approximately 80 initial scripts derived from authorized educational materials would guarantee the highest level of functional validity and domain relevance. These scripts were compiled into a programmatic representation (a Python dictionary) and then converted into the JSONL format. This structured output was essential for providing clear context and desired output constraints to the generative model, enabling efficient batch processing;
2. LLM-Assisted Augmentation: The core augmentation was executed using a custom Python script interfacing with the Anthropic API, leveraging the high-level reasoning and detail tracking capabilities of the Claude Opus 4.1 model. Scripts were sent in batches to the model, which was instructed to generate analogous, yet distinct Shelly scripts while rigorously preserving the characteristic coding style and underlying device concepts observed in the original corpus. This augmentation strategy ensured comprehensive coverage of device-specific components [6];
3. Execution-Based Validation: Every one of the synthetically generated scripts was subjected to rigorous execution-based validation. Each script was manually tested separately on its corresponding Shelly device [7]. Scripts exhibiting functional defects were either manually corrected to align with proper device behavior or excluded from the training corpus when correction was impractical. This critical process established functional correctness essential for the training of the model.

4. Model Training and Fine-Tuning

For evaluation purposes, the Seed-Coder-8B-Instruct-bnb-4bit model was fine-tuned using Low-Rank Adaptation (LoRA) [8] on the synthetic dataset.

The training was conducted using 1 NVIDIA V100 GPU with 32GB on the premises of Paisii Hilendarski University of Plovdiv, Faculty of Mathematics and Informatics. LoRA enables parameter-efficient fine-tuning by updating only a small fraction of model parameters, making it practical for academic resource constraints. The 4-bit quantized base model further reduces memory requirements while maintaining adequate performance for domain-specific adaptation.

5. Evaluation: Base Model vs. Fine-Tuned Model Comparison

To validate the effectiveness of our synthetic dataset, a comparative evaluation was conducted between the base instruct model and the LoRA-adapted model fine-tuned on the created synthetic Shelly dataset. Both models were evaluated on a standard test task: "Control air conditioning and dehumidifier based on temperature and humidity data with thresholds", representing a common real-world automation scenario.

The evaluation focused on three critical criteria:

1. Code Length and Resource Efficiency: Given the memory constraints of the Shelly devices (ESP32-based with limited RAM and flash storage), the generated script must be concise while maintaining functionality;
2. Correct Usage of Shelly API: Proper usage of methods, component references and adherence to JavaScript Espruino syntax constraints, handling HTTP requests to external weather APIs, JSON response parsing and error handling for network operations;
3. Functional Completeness: Coverage of the specified requirements including device control based on thresholds from external data.

5.1. Base Model Output Characteristics

The instruct model produces a verbose solution demonstrating general programming competence. However, the model revealed critical gaps in domain-specific knowledge and limitations for IoT deployment. On one hand, the code was significantly longer than necessary for the constrained Shelly devices, potentially approaching memory limits. The script could have been implemented much more concisely while maintaining full functionality. More critically, the model generated incorrect API methods names, demonstrating the hallucination problem that motivated this study. While the HTTP request handling was correct, the fundamental error in the scheduling mechanism would prevent the script from functioning on an actual Shelly device and would require manual debugging and correction by a person familiar with the documentation of

Shelly's API, defeating the purpose of code generation for non-expert users.

5.2. Fine-Tuned Model Output Characteristics

In contrast, the LoRA-adapted model fine-tuned with our synthetic dataset produces correct, concise, and optimized code demonstrating comprehensive domain knowledge. The code length was optimized for the resource limited devices through recognizing the Shelly-specific patterns that enable more direct implementation. Most significantly, the model generated syntactically correct code on the first attempt, meaning further debugging would not be required and the code is ready for deployment on Shelly devices.

6. Future Directions

Several promising directions emerge from this work:

- Chain-of-Thought Reasoning Integration: Implementing a Chain-of-Thought reasoning model [9] could enhance the model's capability of handling complex automation scenarios. By training the model to generate intermediate reasoning steps before producing final code, we could improve explainability and debugging capabilities while potentially improving correctness in sophisticated tasks;
- Dataset Expansion and Model Retraining: Systematic expansion would provide broader device coverage for recently and newly released in the future Shelly devices with novel capabilities, advanced scenario complexity, and improved edge-case coverage. Iterative retraining with expanded datasets would evaluate whether performance improvements plateau or continue to scale effectively [10];
- Performance Benchmarking Against General-Purpose LLMs: A comprehensive benchmarking study would identify specific weaknesses of general-purpose models versus fine-tuned models using functional metrics such as API accuracy, code efficiency, execution performance and scenario coverage.

7. Conclusion

This work presents a methodology for addressing data scarcity in specialized IoT code generation through systematic synthetic data creation, execution-based hardware validation and fine-tuning. Starting from 80 verified Shelly automation scripts, a dataset of 1,100 functional examples was successfully generated and validated. The evaluation demonstrates that the fine-tuned model produces significantly more concise and hardware-appropriate code compared to

general-purpose models. This current methodology paves the way for a potential framework to be used for developing specialized code generation models in niche domains, characterized by limited public code repositories or proprietary APIs.

Acknowledgments

This work was supported by Paisii Hilendarski University of Plovdiv, Faculty of Mathematics and Informatics, through project SP25-FMI-008 "Research and implementation of modern language models and artificial neural networks for automated processing, forecasting, and structuring of data and texts in specific application domains". The authors thank Shelly Group for providing the required devices, essential for validation, as well as Shelly Academy, powered by SoftUni Global for the high-quality learning materials and code examples that served as the foundation of the seed corpus.

References

[1] S. Kim, J. Suk, X. Yue, V. Viswanathan, S. Lee, Y. Wang, K. Gashteovski, C. Lawrence, S. Welleck, G. Neubig, Evaluating Language Models as Synthetic Data Generators, *Proc. of the 63rd Annual Meeting of the Association for Computational Linguistics*, 2025, Vol. 1, pp. 6385–6403, https://aclanthology.org/2025.acl-long.320.pdf

[2] Z. Zhang, C. Wang, Y. Wang, E. Shi, Y. Ma, W. Zhong, J. Chen, M. Mao, Z. Zheng, LLM Hallucinations in Practical Code Generation: Phenomena, Mechanism, and Mitigation, *Proc. of the ACM on Software Engineering*, 2025, pp. 481–503, ISSN: 2994-970X, https://dl.acm.org/doi/pdf/10.1145/3728894

[3] A. Englhardt, J. Klamroth, P. Hirsch, A. Mueller, K. Rasheed, S. Shastri, D. Jain, K. Li, A. Kamath, F. Tschopp, Exploring and Characterizing Large Language Models For Embedded System Development and Debugging, *Proc. of the 21st ACM Conference on Embedded Networked Sensor Systems*, 2023, pp. 624–637, arXiv:2307.03817, https://arxiv.org/pdf/2307.03817

[4] S. Monov, N. Pavlov, D. Trifonova, BGT5 – Pre-trained T5 Model on Bulgarian Data, *International Journal of Differential Equations and Applications*, 2024, Vol. 23, No. 1, pp. 141–149, https://www.researchgate.net/publication/387787088_BGT5_-_PRE-TRAINED_T5_MODEL_ON_BULGARIAN_DATA

[5] H. Yang, M. Li, M. Han, Z. Li, W. Xu, EmbedGenius: Towards Automated Software Development for Generic Embedded IoT Systems, 2024, arXiv

preprint arXiv:2412.09058, https://arxiv.org/pdf/2412.09058

[6] M. Nadăş, L. Dioşan, A. Tomescu, Synthetic Data Generation Using Large Language Models: Advances in Text and Code, 2025, arXiv preprint arXiv:2503.14023, https://arxiv.org/pdf/2503.14023

[7] Z. Xu, Y. Liu, Y. Yin, M. Zhou, R. Poovendran, KodCode: A Diverse, Challenging, and Verifiable Synthetic Dataset for Coding, 2025, arXiv preprint arXiv:2503.02951, https://arxiv.org/pdf/2503.02951

[8] E. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, LoRA: Low-Rank Adaptation of Large Language Models, *Proc. of the Tenth International Conference on Learning Representations*, 2022, arXiv preprint arXiv:2106.09685, https://arxiv.org/pdf/2106.09685v1/1000

[9] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models, *Advances in neural information processing systems*, 2022, pp. 24824–24837, ISBN: 9781713871088, https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf

[10] Z. Qin, Q. Dong, X. Zhang, L. Dong, X. Huang, Z. Yang, M. Khademi, D. Zhang, H. Awadalla, Y. Fung, W. Chen, M. Cheng, F. Wei, Scaling Laws of Synthetic Data for Language Models, 2025, arXiv preprint arXiv:2503.19551, https://arxiv.org/pdf/2503.19551

Simeon Monov[1], Mariana Veselinova[1]
[1] Paisii Hilendarski University of Plovdiv,
Faculty of Mathematics and Informatics,
236 Bulgaria Blvd., 4027 Plovdiv, Bulgaria
Corresponding author: smonov@uni-plovdiv.bg