Towards Automated Embedded Systems Programming

Imam Nur Bani Yusuf
School of Computing and Information System
Singapore Management University
imamy.2020@phdcs.smu.edu.sg

***Abstract*—Writing code for embedded systems poses unique challenges due to hardware involvement. Developers often need to learn domain-specific knowledge to write embedded codes. Learning such knowledge is time-consuming and hinders developers' productivity. This paper presents a proposal for an automated code generation approach, specifically designed for embedded systems. The work is composed of three milestones, i.e., understanding the needs of embedded developers by analyzing posts from discussion forums, developing a tool to recommend driver libraries of I/O hardware and generate its interface configurations and usage patterns, and improving the generation accuracy of the prior tool using program analysis techniques. The tool will be evaluated using various metrics from machine translation, classification, and information retrieval fields.**

***Index Terms*—code generation, library recommendation, embedded system, hardware configuration**

I. Motivation and Research Problem

An embedded system is a system that integrates software and hardware to achieve a specific task. An embedded system can be decomposed into the code, I/O (Input/Output) hardware, and embedded controller. The code contains the logic that represents the functionality, such as reading sensor values or moving actuators. The controller controls a set of I/O hardware to interact with the environment, based on the logic of the code. There are various embedded system use cases, ranging from mission-critical (e.g., autonomous car) to massive applications (e.g., smart home).

Writing code for embedded systems poses unique challenges due to hardware involvement [1]–[3]. Makhshari and Mesbah [1] find that writing codes for embedded systems often require diverse background knowledge besides coding. Consider the example in Fig. 1. A simple action to move the servo using the *write* method requires the developer to know how to correctly configure interfaces/pin numbers when calling the *attach* method. The developer may not be familiar with such domain-specific knowledge [3]. Learning such domain-specific knowledge is possible, but it can hinder the productivity of a developer. One potential solution to lowering the learning curve of such domain-specific knowledge is to develop an automated code generation tool based on deep learning techniques. Recent studies on automated code generation have shown that deep learning-based approaches can yield promising results [4]. There are a variety of input specifications that have been used, such as natural language descriptions, codes, or even pixels [5], [6].

Based on the literature study, I identify two research gaps. First, the programming needs of developers when writing embedded codes are still unclear. No prior studies have scrutinized how embedded codes are written and how I/O hardware is controlled in the code. Second, as a consequence of the first gap, there is no tool available to help developers write embedded codes. Hence, the research questions of this study are: 1) to understand the programming needs of developers when writing embedded codes and 2) to develop a tool that can help developers easily write embedded codes without worrying about the required domain-specific knowledge.

```
Servo myServo;
myServo.attach(pin=10);
myServo.write(10)
myServo.detach()
```

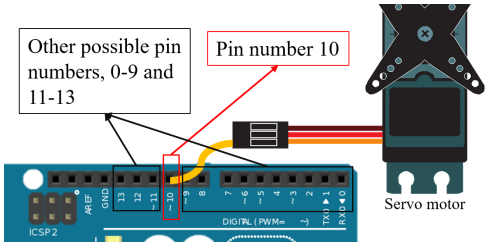

Fig. 1. An example of how the setup in the physical and code spaces should match. The servo should be connected to the interface number 10 in the physical space and the Servo object must be initialized using pin number 10 in the code space.

II. Expected Contributions

The main objective of my study is to understand the common needs of embedded developers and propose a tool to help developers write embedded codes based on such needs. I divide my work into three milestones.

1) Scrutinizing the common needs of developers when writing embedded codes. The output of the first milestone is several insights into the developers' needs when writing embedded codes. Such insights will be used to justify the design of the proposed tool for the subsequent milestone.
2) Exploring the possibility of automatically generating embedded codes using natural language descriptions that satisfy the common needs of embedded developers. The output of the second milestone is a deep learning-based tool that produces hardware interface configurations and API usage patterns (without parameters) of individual I/O hardware.

3) Improving the tool proposed in the second milestone to generate complete embedded codes, not only the interface configurations and API usage patterns of individual I/O hardware. The output of the third milestone is a deep learning-based tool that generates complete embedded codes.

III. Related Works

Several empirical studies [1]–[3] related to developing embedded systems have been conducted. Makhshari and Mesbah [1] found that hardware-specific knowledge causes embedded development in the Internet of Things (IoT) setting difficult. According to Uddin et al. [2], microcontroller configuration is one of the three most frequently discussed IoT-related topics in StackOverflow. Booth et al. [3] conducted a user study and shows that 80% of circuit-related problems are caused by miswiring and missing electronic components.

Some works have been proposed to recommend relevant library usage patterns [7]–[15] given a natural language query. However, generating only library usage patterns is insufficient because developers also need to configure I/O hardware. Another line of works is to develop a tool to allow developers to configure interfaces using a specialized hardware [16]–[18]. However, the specialized hardware may not be available for everyone. Moreover, a block-based programming [19] approach and interactive tutorials [20]–[23] have been developed. However, such approaches are not scalable because they require hand-crafted templates.

IV. Proposed Approach

I describe the possible approaches for achieving each milestone as follows.

Common needs of developers. I plan to gather and analyze developers' questions from various embedded system discussion forums. Some examples of the forums are Arduino Discussion Forum [24] and Raspberry Pi Forum [25]. First, the questions are crawled using several criteria, such as whether the questions contain solutions, codes, or several predefined keywords. Defining the criteria may require domain-specific knowledge. Second, the questions are clustered using topic modeling techniques to get a high-level summary of the questions. Third, some questions from each topic are sampled and each sampled question is manually inspected to extract the insights.

Recommending interface configurations and API usage patterns. I plan to develop a tool on top of existing large language models to generate hardware configurations and API usage patterns given natural language descriptions. First, the tool identifies I/O hardware in the description. Second, the tool recommends relevant driver libraries for each I/O hardware. Third, the tool generates the interface configurations and API usage patterns of each recommended library. Three aspects need to be carefully designed when developing a tool that utilizes deep learning: the neural network architecture, the input and output representation, and the transfer learning mechanism. Transformer [26] is a strong architecture candidate because prior studies [4], [14], [15], [27] have demonstrated that Transformer can achieve state-of-the-art performance on various coding tasks. Some candidates for the input representation are the question title, question body, and code in the question. The outputs of the model are the interface configurations and the API usage patterns (without parameters) from the driver libraries. With the rise of language models with huge parameters such as BLOOM [28], one suitable candidate for the transfer learning mechanism is prompt-tuning.

Generating complete embedded codes. I plan to improve the proposed tool in the second milestone by combining the tool with program analysis techniques. Specifically, the goal of the third milestone is to arrange the generated recommendations into a complete embedded code using program analysis techniques. However, there are two challenges. The first challenge is generating the correct API parameters can be difficult due to the diversity of the parameters; API parameters can be an arbitrary value specified by a developer. I plan to use domain-specific heuristics and pattern mining techniques to tackle the parameter diversity challenge. The second challenge is writing the functional logic of the complete code (e.g., if the temperature is more than x then do y) may not be straightforward, especially if the input description involves multiple hardware. One idea is to study the control flow between multiple hardware in codes and use heuristics or develop some techniques to generate the functional logic to form the complete code.

V. Future Plans

Evaluation plan. To evaluate the quality of the recommended library, I plan to use classification metrics (e.g., precision, recall, and F1) and also retrieval metrics (e.g., Normalized Discounted Cumulative Gain or NDCG [29]). For evaluating the code generation performance, I plan to leverage several machine translation metrics, such as BLEU [30], METEOR [31], and ROUGE [32] scores. BLEU score is the hits of n-grams of translation results with its ground truths. ROUGE score reflects the number of n-grams in the ground truths that appear in the results. METEOR measures the relevancy based on the harmonic mean of unigram precision and recall.

Completion timeline. For milestone (1), the expected completion time is five months. I am still in the phase of collecting forum posts from Arduino Discussion Forum [24]. The next steps after the data collection are data cleaning, data labeling, and data analysis. Data cleaning may require one month, while data labeling may require three months. Another month is for data analysis. On the other hand, I plan to finish milestone (2) in three months because I have started the initial development of the tool. For milestone (3), I plan to finish in six months. My expected graduation time is by the end of next year, and there are still a few months after I finish milestone (3) until the end of the year. The remaining time will be used for any possible iterations due to resubmissions.

REFERENCES

[1] A. Makhshari and A. Mesbah, "Iot bugs and development challenges," in *International Conference on Software Engineering*. IEEE, 2021, pp. 460–472. [Online]. Available: https://doi.org/10.1109/ICSE43902.2021.00051

[2] G. Uddin, F. Sabir, Y. Guéhéneuc, O. Alam, and F. Khomh, "An empirical study of iot topics in iot developer discussions on stack overflow," *Empir. Softw. Eng.*, vol. 26, no. 6, p. 121, 2021. [Online]. Available: https://doi.org/10.1007/s10664-021-10021-5

[3] T. Booth, S. Stumpf, J. Bird, and S. Jones, "Crossed wires: Investigating the problems of end-user developers in a physical computing task," in *Conference on Human Factors in Computing Systems*. ACM, 2016, pp. 3485–3497. [Online]. Available: https://doi.org/10.1145/2858036.2858533

[4] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "Codebert: A pre-trained model for programming and natural languages," in *EMNLP*, ser. Findings of ACL, vol. EMNLP 2020. Association for Computational Linguistics, 2020, pp. 1536–1547. [Online]. Available: https://doi.org/10.18653/v1/2020.findings-emnlp.139

[5] T. Beltramelli, "pix2code: Generating code from a graphical user interface screenshot," in *EICS*. ACM, 2018, pp. 3:1–3:6. [Online]. Available: https://doi.org/10.1145/3220134.3220135

[6] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. B. Clement, D. Drain, D. Jiang, D. Tang, G. Li, L. Zhou, L. Shou, L. Zhou, M. Tufano, M. Gong, M. Zhou, N. Duan, N. Sundaresan, S. K. Deng, S. Fu, and S. Liu, "Codexglue: A machine learning benchmark dataset for code understanding and generation," in *NeurIPS Datasets and Benchmarks*, 2021.

[7] L. Cai, H. Wang, Q. Huang, X. Xia, Z. Xing, and D. Lo, "BIKER: a tool for bi-information source based API method recommendation," in *European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2019, pp. 1075–1079. [Online]. Available: https://doi.org/10.1145/3338906.3341174

[8] M. M. Rahman, C. K. Roy, and D. Lo, "RACK: automatic API recommendation using crowdsourced knowledge," in *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*. IEEE Computer Society, 2016, pp. 349–359. [Online]. Available: https://doi.org/10.1109/SANER.2016.80

[9] Q. Huang, X. Xia, Z. Xing, D. Lo, and X. Wang, "API method recommendation without worrying about the task-api knowledge gap," in *Automated Software Engineering*. ACM, 2018, pp. 293–304. [Online]. Available: https://doi.org/10.1145/3238147.3238191

[10] W. Yuan, H. H. Nguyen, L. Jiang, Y. Chen, J. Zhao, and H. Yu, "API recommendation for event-driven android application development," *Inf. Softw. Technol.*, vol. 107, pp. 30–47, 2019. [Online]. Available: https://doi.org/10.1016/j.infsof.2018.10.010

[11] W. Yuan, H. H. Nguyen, L. Jiang, and Y. Chen, "Libraryguru: API recommendation for android developers," in *International Conference on Software Engineering: Companion Proceeedings*. ACM, 2018, pp. 364–365. [Online]. Available: https://doi.org/10.1145/3183440.3195011

[12] F. Thung, D. Lo, and J. Lawall, "Automated library recommendation," in *Working Conference on Reverse Engineering*. IEEE Computer Society, 2013, pp. 182–191. [Online]. Available: https://doi.org/10.1109/WCRE.2013.6671293

[13] F. Thung, S. Wang, D. Lo, and J. Lawall, "Automatic recommendation of API methods from feature requests," in *Automated Software Engineering*. IEEE, 2013, pp. 290–300. [Online]. Available: https://doi.org/10.1109/ASE.2013.6693088

[14] I. N. B. Yusuf, L. Jiang, and D. Lo, "Accurate generation of trigger-action programs with domain-adapted sequence-to-sequence learning," in *International Conference on Program Comprehension*. ACM, 2022, pp. 99–110. [Online]. Available: https://doi.org/10.1145/3524610.3527922

[15] I. N. B. Yusuf, D. B. A. Jamal, L. Jiang, and D. Lo, "Recipegen++: an automated trigger action programs generator," in *European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, A. Roychoudhury, C. Cadar, and M. Kim, Eds. ACM, 2022, pp. 1672–1676. [Online]. Available: https://doi.org/10.1145/3540250.3558913

[16] W. Lee, R. Prasad, S. Je, Y. Kim, I. Oakley, D. Ashbrook, and A. Bianchi, "Virtualwire: Supporting rapid prototyping with instant reconfigurations of wires in breadboarded circuits," in *Tangible, Embedded, and Embodied Interaction*, 2021, pp. 4:1–4:12. [Online]. Available: https://doi.org/10.1145/3430524.3440623

[17] Y. Kim, H. Lee, R. Prasad, S. Je, Y. Choi, D. Ashbrook, I. Oakley, and A. Bianchi, "Schemaboard: Supporting correct assembly of schematic circuits using dynamic in-situ visualization," in *User Interface Software and Technology*. ACM, 2020, pp. 987–998. [Online]. Available: https://doi.org/10.1145/3379337.3415887

[18] T. Wu, B. Wang, J. Lee, H. Shen, Y. Wu, Y. Chen, P. Ku, M. Hsu, Y. Lin, and M. Y. Chen, "Circuitsense: Automatic sensing of physical circuits and generation of virtual circuits to support software tools," in *User Interface Software and Technology*, 2017, pp. 311–319. [Online]. Available: https://doi.org/10.1145/3126594.3126634

[19] F. Anderson, T. Grossman, and G. W. Fitzmaurice, "Trigger-action-circuits: Leveraging generative design to enable novices to design and build circuitry," in *User Interface Software and Technology*, 2017, pp. 331–342. [Online]. Available: https://doi.org/10.1145/3126594.3126637

[20] Y. Kim, Y. Choi, D. Kang, M. Lee, T. Nam, and A. Bianchi, "Heyteddy: Conversational test-driven development for physical computing," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 3, no. 4, pp. 139:1–139:21, 2019. [Online]. Available: https://doi.org/10.1145/3369838

[21] J. Warner, B. Lafreniere, G. W. Fitzmaurice, and T. Grossman, "Electrotutor: Test-driven physical computing tutorials," in *User Interface Software and Technology*. ACM, 2018, pp. 435–446. [Online]. Available: https://doi.org/10.1145/3242587.3242591

[22] J. U. Davis, J. Gong, Y. Sun, P. K. Chilana, and X. Yang, "Circuitstyle: A system for peripherally reinforcing best practices in hardware computing," in *User Interface Software and Technology*. ACM, 2019, pp. 109–120. [Online]. Available: https://doi.org/10.1145/3332165.3347920

[23] D. Drew, J. L. Newcomb, W. McGrath, F. Maksimovic, D. Mellis, and B. Hartmann, "The toastboard: Ubiquitous instrumentation and automated checking of breadboarded circuits," in *User Interface Software and Technology*. ACM, 2016, pp. 677–686. [Online]. Available: https://doi.org/10.1145/2984511.2984566

[24] "Arduino forum," https://forum.arduino.cc/, (Accessed on 11/18/2022).

[25] "Raspberry pi forums," https://forums.raspberrypi.com/, accessed: 2023-03-18.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, 2017, pp. 5998–6008.

[27] Y. Wang, W. Wang, S. R. Joty, and S. C. H. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Empirical Methods in Natural Language Processing*, 2021, pp. 8696–8708. [Online]. Available: https://doi.org/10.18653/v1/2021.emnlp-main.685

[28] T. L. Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilic, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon, M. Gallé, J. Tow, A. M. Rush, S. Biderman, A. Webson, P. S. Ammanamanchi, T. Wang, B. Sagot, N. Muennighoff, A. V. del Moral, O. Ruwase, R. Bawden, S. Bekman, A. McMillan-Major, I. Beltagy, H. Nguyen, L. Saulnier, S. Tan, P. O. Suarez, V. Sanh, H. Laurençon, Y. Jernite, J. Launay, M. Mitchell, C. Raffel, A. Gokaslan, A. Simhi, and et al., "BLOOM: A 176b-parameter open-access multilingual language model," *CoRR*, vol. abs/2211.05100, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2211.05100

[29] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of IR techniques," *ACM Trans. Inf. Syst.*, vol. 20, no. 4, pp. 422–446, 2002. [Online]. Available: http://doi.acm.org/10.1145/582415.582418

[30] K. Papineni, S. Roukos, T. Ward, and W. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. ACL, 2002, pp. 311–318. [Online]. Available: https://aclanthology.org/P02-1040/

[31] S. Banerjee and A. Lavie, "METEOR: an automatic metric for MT evaluation with improved correlation with human judgments," in *Proceedings of the Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization@ACL*. Association for Computational Linguistics, 2005, pp. 65–72. [Online]. Available: https://aclanthology.org/W05-0909/

[32] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://aclanthology.org/W04-1013