Developing RAGs for robot code generation

Downloaded from: https://research.chalmers.se, 2026-09-14 21:33 UTC

Citation for the original published paper (version of record):
Salunkhe, O., Chen, S., Syberfeldt, A. et al (2026). Developing RAGs for robot code generation. IOP Conference Series: Materials Science and Engineering, 1342.
http://dx.doi.org/10.1088/1757-899X/1342/1/012064

N.B. When citing this work, cite the original published paper.

research.chalmers.se offers the possibility of retrieving research publications produced at Chalmers University of Technology. It covers all kind of research output: articles, dissertations, conference papers, reports etc. since 2004. research.chalmers.se is administrated and maintained by Chalmers Library

(article starts on next page)

IOP Conference Series:
Materials Science and
Engineering

PURPOSE-LED
PUBLISHING™

PAPER • **OPEN ACCESS**

Developing RAGs for robot code generation

To cite this article: Omkar Salunkhe *et al* 2026 *IOP Conf. Ser.: Mater. Sci. Eng.* **1342** 012064

View the article online for updates and enhancements.

You may also like

- VISION: a modular AI assistant for natural human-instrument interaction at scientific user facilities
Shray Mathur, Noah van der Vleuten, Kevin G Yager et al.
- ChemLit-QA: a human evaluated dataset for chemistry RAG tasks
Geemi P Wellawatte, Huixuan Guo, Magdalena Lederbauer et al.
- The evolution of ChatGPT for programming: a comparative study
Rushabh Jain, Jyoti Thanvi and Akila Subasinghe

This content was downloaded from IP address 129.16.68.212 on 01/06/2026 at 11:05

IOP Conf. Series: Materials Science and Engineering **1342** (2026) 012064 doi:10.1088/1757-899X/1342/1/012064

Developing RAGs for robot code generation

Omkar Salunkhe[1]*, Siyuan Chen[1], Anna Syberfeldt[1,2] and Johan Stahre[1]

[1] Department of Mechanical Engineering, Chalmers University of Technology, Gothenburg, Sweden
[2] School of Engineering, University of Skövde, Skövde, Sweden

*E-mail: omkar.salunkhe@chalmers.se

Abstract. The emergence of generative AI marks a transformative shift in industrial automation. Traditional robot programming relies on manually written, low-level code that requires specialised expertise, limiting flexibility and accessibility. Recent advances in Large Language Models (LLMs) such as ChatGPT and Mistral introduce new paradigms for automated code generation. However, concerns about data security, model hallucinations, and the opaque reasoning of generative systems continue to hinder their adoption in industry. A promising approach to address these challenges is Retrieval-Augmented Generation (RAG), where the generative model draws on curated, domain-specific data sources controlled by the user. By combining structured knowledge retrieval with generative inference, RAG-based systems can produce robot code that is not only more accurate and context-aware but also verifiable and transparent. This approach enhances user trust and enables safer integration of AI in industrial settings.

This paper explores the application of Retrieval-Augmented Generation (RAG)-based architectures - a method that combines information retrieval with LLMs - for robot code generation. RAG-based systems enable LLMs to access and utilise domain-specific data, thereby grounding their outputs in reliable knowledge. By leveraging these techniques, robotics developers can achieve more accurate and efficient code generation, potentially accelerating innovation in autonomous systems. Furthermore, it presents a conceptual framework for RAG-enhanced robot programming that balances autonomy with human oversight. The proposed framework enhances the adaptability and intelligence of automated programming by providing a transparent, controllable, and explainable alternative to conventional AI-driven methods, paving the way for more reliable and human-centric automation in future manufacturing environments.

1. Introduction

The rapid development of Generative Artificial Intelligence (GenAI) is currently transforming how industrial robots are programmed [1]. Traditional robot programming usually depends on vendor-specific tools, proprietary languages, and skilled programmers. Conventional methods are time-consuming and limit scalability, especially in today's manufacturing environments where product variation and customisation demands are increasing. GenAI, particularly through Large Language Models (LLMs), introduces an alternative paradigm for robot programming [2]. GenAI

Content from this work may be used under the terms of the Creative Commons Attribution 4.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.
Published under licence by IOP Publishing Ltd

can generate and refine robot code using natural language prompts, lowering the barrier to program industrial robots and enabling non-experts to carry out these programming tasks. There is a great potential to utilise LLMs' capacity for pattern recognition and contextual understanding in robot programming, but so far, their use in real industrial contexts remains limited. Issues such as model hallucination, lack of explainability, and data privacy concerns hinder the reliability and trustworthiness of AI-generated robot code [1].

Retrieval-Augmented Generation (RAG) is a promising solution that merges the creative capabilities of generative models with factual, domain-specific knowledge [3]. In a RAG setup, a generative model is linked to a retrieval system that sources relevant information from internal databases such as robot manuals, code repositories, and various types of process documentation [4]. By leveraging RAG architectures in robot programming, it is possible to achieve secure, explainable, and dependable code that aligns with company safety and quality requirements in manufacturing. Additionally, RAG frameworks facilitate adaptive learning for different robot models and programming tasks, promoting quicker deployment.

This paper investigates the development of RAG systems for generating robot code, with an emphasis on improving accuracy, interpretability, and data security in within industrial settings. The authors propose a conceptual framework and system architecture designed for robot code generation using IP-protected data. Initial implementation and evaluation highlight the approach's potential to improve confidence and trust in AI-assisted robot programming, paving the way for improved collaboration between humans and robots in manufacturing environments.

2. Background

2.1 Robot programming

Traditionally, robots are programmed using offline software provided by robot vendors such as RobotStudio by ABB, MotoSim by Yaskawa Motoman, and Kuka.Sim by Kuka, to name a few. This programming approach is a relatively manual and specialised process. These vendor-specific software programs require in-depth knowledge of syntax, coordinate systems, and motion-control logic, which is proprietary to each vendor. Offline or online programming for robots requires substantial effort. These tasks can be time-consuming and costly, especially if there is a minor change in part geometry, end-effector configuration, or integration of new sensors or fixtures, which can take hours or days of code adjustments and validation [5]. Robot programming has also been the domain of highly skilled specialists, which can contribute to long deployment cycles and high integration costs, particularly in automotive manufacturing, where product variation is high, and the constant need to reduce production costs is constant.

2.2 Generative AI in robotics

Generative AI powered by large language models such as ChatGPT, Claude, and Gemini is demonstrating impressive capabilities for code generation. Research shows that AI-assisted code generation based on LLMs achieves code accuracy of upwards of 90% [6]. The industry is also witnessing increased interest in low-code/no-code robot programming solutions aimed at eliminating traditional programming. Large language models are successfully applied to generate robot and PLC code, with iterative refinement and error-correction mechanisms to improve code quality [7, 8]. Despite these advancements, challenges remain in the use of GenAI in robotics,

especially code generation based on domain-specific knowledge, as dataset curation and fine-tuning are resource-intensive processes [9].

2.3 Challenges with LLMs

While the integration of generative AI into robotic programming is promising, some challenges need to be addressed for LLMs to be reliably deployed. A key challenge is the *risk of hallucination*, which means that the LLMs generate believable but incorrect code [10]. In industrial robot programming, hallucinations can manifest as invalid parameters or incorrect task sequences, leading to serious operational consequences and even dangerous situations. RAG systems can further increase hallucination errors if the retrieved documents are outdated or inaccurate. Ensuring that the LLM-generated output is correct is a critical technical challenge. Another key challenge is the *lack of transparency and explainability* in LLMs [11]. When LLMs generate robot code, it is often unclear which sources or training data formed the output. This lack of traceability complicates critical processes such as validation, debugging, and certification processes that are vital for safety-critical industrial robot applications. Maintaining the quality and reliability of generated code continues to pose significant challenges. Therefore, there is a clear need for supervisory mechanisms to monitor and validate code produced by LLMs [12]. Persistent concerns regarding the accuracy and interpretability of LLM-generated code may affect trust and hinder the adoption of these technologies in critical contexts [13]. Without robust mechanisms to verify both the reliability and provenance of generated code, users may lack confidence in its deployment [14].

Data privacy and intellectual property (IP) protection are significant concerns for companies. [15]. Companies have proprietary process data, CAD files, source code, etc., that cannot be shared outside the organisation. When LLMs are used in RAG frameworks that connect to external platforms/APIs, there is a risk that the model may expose or store proprietary information. This is especially problematic for manufacturing companies where production setups, control algorithms, equipment configurations, etc., constitute core competitive assets. As a result, industrial companies often require locally deployed LLMs, potentially combined with other security measures such as encryption. Another challenge is the *responsibility for the output* produced by LLMs [16]. Who is responsible when automatically generated code leads to unintended behaviour of the robot or even accidents? This is an open question with legal and ethical perspectives.

2.4 RAG architectures

The fundamental concept of RAG is to enhance LLMs' generative capabilities by incorporating a retrieval component that accesses an external knowledge base [17]. Customised RAG systems can incorporate domain-specific knowledge, which is crucial for generating accurate and contextually relevant RAPID code. By integrating specialised datasets and knowledge bases, these systems ensure that the generated code aligns with the specific requirements and standards of the domain [3]. When a user provides a query, for example, generating an assembly sequence, the retriever searches a collection of relevant documents and sources of information. This could, for example, be code repositories, robot manuals, controller libraries or CAD models. The retrieved data is inserted into the model's input prompt, thereby augmenting the context available for generation. The LLM then generates code based on a combination of its internal pretraining and the retrieved data. As long as the retrieved data is up to date, this architecture increases the accuracy and domain relevance of the generated code [18].

State-of-the-art RAG architectures extend beyond the basic structure by including multiple components that are specialised for industrial applications. A typical setup consists of a semantic embedding stored in a vector database, which indexes process documents, robot APIs, and safety guidelines. Retrieved segments are filtered through re-ranking mechanisms that prioritise domain relevance, for example, selecting data corresponding to the correct robot model or control library version [17]. The final prompt is generated by combining the user query with the retrieved information, often after transforming it into a format the LLM can efficiently process. In some systems, the generated code is passed through verification layers, such as rule-based validators or digital twin simulations, to ensure compliance with safety standards and operational constraints [18].

3. Research Methodology

The research study follows the Design Science Research Methodology (DSRM) as described by Peffers et al. [19], grounded in the design science principles formulated by Hevner et al. [20]. DSRM is appropriate for this study, as the primary contribution is the design and development of a novel artefact (the RAG system) intended to address an identified industrial problem. The research problem was identified through an analysis of current industrial robot programming practices and recent advances in LLMs. In line with the problem-centred nature of design science research, this study aims to develop a solution that improves the accuracy, transparency, and trustworthiness of AI-generated robot code by grounding generation in curated, domain-specific knowledge sources. Following the DSRM process, the artefact, i.e. The RAG system, is designed and developed iteratively. The artefact is demonstrated through representative robot programming tasks and is evaluated qualitatively by comparing its outputs with those of a general-purpose LLM. Evaluation criteria focus on accuracy and context-awareness, in line with the relevance-driven evaluation principles of design science research.

4. Proposed RAG System and Architecture

A summary of the differences between customised RAG and general-purpose LLMs is presented in Table 1. The use case presented in this paper proposes a RAG system that uses IP-protected documents to generate RAPID code for running ABB robots. While general-purpose LLMs like ChatGPT generate robot code primarily based on pre-training, the proposed RAG system fundamentally differs from such an approach. Pre-training LLMs do not have vendor-specific knowledge, such as the latest naming convention used in RAPID. RAG systems provide the ability to access relevant, up-to-date information from the specific knowledge database in the RAPID manual and technical documentation. Often, companies bake in their own standards and syntaxes that are not accessible to general-purpose LLMs but can be used in RAG systems to generate relevant and valuable answers. This ensures that code generation is aligned with the proper syntax and company standards.

While LLMs provide opaque, incorrect answers, the RAG system shows which documents were retrieved, which code snippets were used, and how the final answer was constructed, allowing traceability and transparency. Hallucinations in robot code generation constitute a significant problem with general-purpose LLMs, which provide answers that are irrelevant and contain made-up parameters, motions, tool and work-object definitions, as they lack proper context

dependence and domain-specific knowledge. RAG systems reduced these hallucinations as their answer retrieval is limited by the technical documentation they have access to.

The system architecture of our RAG-based code-generation framework for industrial robot programming comprises interconnected layers that work in harmony to transform user queries into syntactically correct, contextually appropriate RAPID code.

Table 1: Customised RAG system vs General LLMs

Aspect	Customised RAG System	General LLMs
Knowledge Source	A combination of LLMs and externally updated databases.	Use of only pre-trained data
Transparency	Full: All steps in generation are visible (e.g., pre-processing, embedding, indexing, retrieval, prompt engineering)	Opaque: sources can be listed, but logic and data flow are inaccessible to the user.
Domain Adaptation	Highly customisable to a specific domain.	Generic
Trustworthiness	High, explainable answers are grounded in retrievable datasets.	Low, high risk of hallucination. Unverified data
Privacy and IP protection	Complete control over data, including storage and retrieval, which ensures IP-protected datasets remain within the organisation.	Limited data control, no guarantee of IP-protected data remaining private.

4.1 Pre-processing

Pre-processing is a critical step in any data-driven AI pipeline. The primary purpose of this step is to transform raw, heterogeneous data into a standardised format that is suitable for downstream processing. In the proposed case, the foundation lies in the data processing layer, which ingests and prepares heterogeneous input sources such as existing robot programs (.tmod, modx), PDF files, technical documentation, and instruction manuals, then converts them into plain text and segments them into manageable chunks. Effective pre-processing ensures the knowledge base is both comprehensive and accessible, reducing data noise and improving the quality of information retrieval. The approach proposed in this paper employs content-aware chunking strategies that differentially process textual documentation through sentence-boundary preservation and code repositories through structural segmentation along module and procedure boundaries. This aligns with best practices in information retrieval and natural language processing, where data segmentation and normalisation serve as the basis for robust search [21].

4.2 Vectorised Database

In a RAG architecture, the vectorised database stores processed knowledge and textual information as a high-dimensional embedding. These embeddings are typically generated by a transformer-based language model, such as BERT or Sentence Transformers, mapping each text segment to a point in a continuous semantic space. This approach builds on advances in representation learning, enabling rapid retrieval of semantically relevant information. Vector databases such as FAISS and Milvus are commonly used for this purpose and are also used in our case, as they provide efficient large-scale similarity search [22, 23]. The RAG system proposed in this paper uses a vector database (such as FAISS) to store embeddings of segmented code files and instructional PDFs.

4.3 User query processing

User query processing is essential for bridging the gap between a user's intent and the machine's understanding of the query. In a typical RAG system, this involves cleaning, normalising, and embedding the user's query so it can be compared with the knowledge base in the same semantic

space. Techniques such as tokenisation and embedded generation, using models such as BERT, are standard processes that ensure the query is contextually represented, enabling accurate retrieval [24]. The proposed RAG system is specifically tailored to the robotics domain; for example, queries for generating robot code are pre-processed to extract relevant intent and technical terms. Domain-specific embedding strategies are used in this RAG system to efficiently map queries about robot programming to the underlying technical documentation and examples in the knowledge base.

4.4 Retrieval phase

The retrieval phase utilises similarity search algorithms (e.g. k-nearest in the vector space) to identify the most relevant knowledge chunks from the vectorised database. This is foundational to RAG systems, as it serves as the basis for a generative model of factual, contextually relevant information [25, 24]. The embedding and retrieval layer generates unified vector representations using advanced language models and implements a dual-path semantic search mechanism. This layer enables simultaneous querying across both documentation and code corpora by measuring cosine similarity in a high-dimensional embedding space. The retrieved contexts are then passed to the prompt engineering Layer, which incorporates specialised templates that can be dynamically selected based on query characteristics and available context. This layer strategically formats the retrieved information, including source metadata and relevance indicators, to guide the generation process. Our proposed RAG system is optimised for technical documents and RAPID robot code snippets. The retrieval logic is fine-tuned to prioritise documents that match both the technical requirements and the operational context of the user's query stored in the Vector databases.

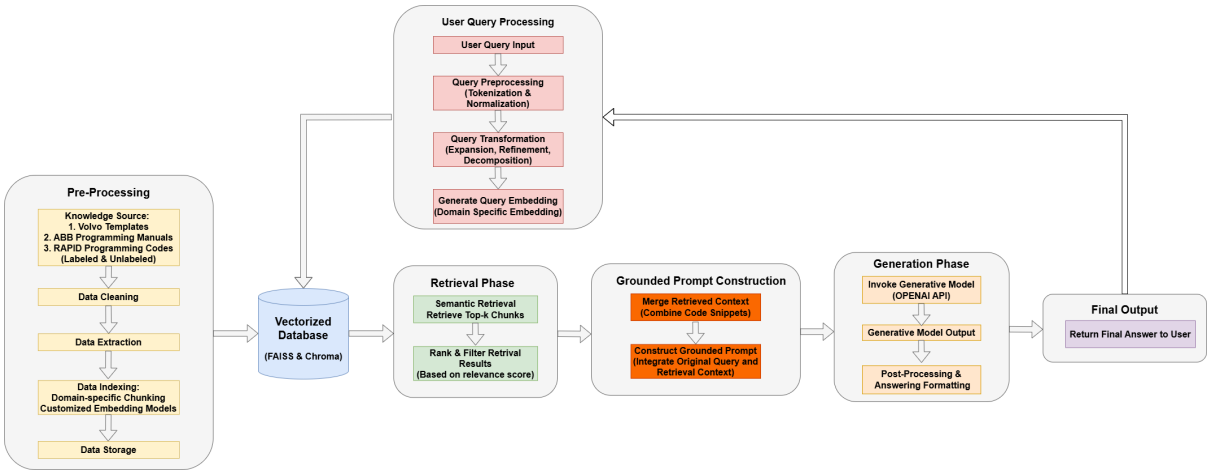

Figure 1: Proposed RAG system architecture

4.5 Grounded prompt generation

The grounded prompt construction combines retrieved context with the user's query into a single input for the generative model. This approach is especially helpful in reducing hallucinations and for improving factual accuracy by explicitly linking the model's output to retrieved information [26]. In the proposed RAG system, the users' code generation query (e.g., "*Generate a RobotStudio RAPID program that picks up a workpiece, at point A and places it at point B.*" is merged with the most relevant robot code snippet and document retrieved from the vector database. The prompt template then guides the model (for example: *"You are an expert RobotStudio RAPID code assistant.*

Use the provided context to generate a complete RAPID program"). This ensures that the output is not only syntax accurate but also functionally applicable in industrial settings.

4.6 Generation Phase

The apex of the architecture is formed by the generation and synthesis Layer, where large language models with carefully calibrated parameters produce the final RAPID code outputs. The entire architecture is orchestrated by a unified pipeline controller that manages the workflow from query intake to result delivery, maintaining comprehensive metadata tracking throughout.

A summary of the main differences between the RAG system and General LLMs is presented in Table 1. These differences serve as the basis for evaluating the proposed RAG system and, in the case of the paper, ChatGPT 4o.

5. Evaluation and analysis

To analyse the effectiveness of the proposed RAG system, a series of tests was conducted using a vectorised knowledge base populated with RAPID code examples and technical documents. These include RAPID for different ABB robot types and processes, predominantly in welding operations. The RAG output was then compared with one generated by ChatGPT for the same query, using the same documents and manuals.

The following three criteria were used to assess the performance of the proposed RAG system and ChatGPT 4o.

- Code Accuracy: Does the generated code match the template and reference structure for RAPID programs?
- Context Awareness: How well do RAG and LLM incorporate task-specific details from the prompt and relevant documents?
- Traceability: How well can each generated code snippet be traced back to the source for verification purposes?

An example of code generation using our proposed RAG vs ChatGPT 4o is presented in Annex A, based on the following query*: "Create a RAPID procedure to move the robot from point A to point B."* The result of the comparison is presented in Table 2.

Table 2: RAG vs LLM evaluation

Criterion	**RAG System Output**	**ChatGPT 4o Output**
Code Accuracy	Matches reference structure, includes all task steps	Generic, omits key steps and details
Context Awareness	Incorporates prompt details and domain knowledge	Lacks task-specific adaptation
Traceability	Each segment is traceable to source documents	No traceability or source references

A summary of RAG output and ChatGPT 4o output is presented in Table 2. Furthermore, the output results were also evaluated for reliability, transparency and hallucination reduction. With respect to Reliability, RAG output generated correct syntax, proper variable declaration and references to tools and work objects. In contrast, ChatGPT's response included invalid work object definitions, incorrect coordinates, and undefined variables (e.g., made-up weldWobj reference), requiring manual check and diminishing trust in the AI-generated answer. In the context of transparency, the customised pipeline based on our RAG provides details on embedding and indexing processes, the utilisation of high-dimensional vector representations, and the FAISS library for similarity

search. This fully transparent process enables tracing data to its origin through clear code and metadata mapping; this cannot be said of ChatGPT's response. Domain-specific RAG provides a systematic, transparent method for constructing prompts for LLMs. The retrieved context is inserted directly into the prompts tailored to the system, in this case, RAPID code for ABB robots. This approach reduces the likelihood of hallucinations and ensures that the models' responses are grounded in verified, domain-specific information. While ChatGPT generated several inaccuracies, including missing tool offsets, non-standard work object definitions and unreferenced positions like pWeld and pHome. While data privacy cannot be directly measurable through generated code, the RAG system safeguards IP-protected robot code and company-specific standards by limiting retrieval to locally stored company-controlled documents. This contrasts with general-purpose LLMs like ChatGPT 4o, which rely on servers in remote locations, often beyond the legal limits of the countries where companies are located, and cannot guarantee the protection of IP-protected documents.

In the proposed RAG system, every step, from data processing and ingestion to generation, is clearly defined, modifiable and auditable, thus enabling reproducibility and traceability. Such an approach supports scientific rigour by ensuring that identical inputs produce identical outputs and that every response can be traced back to its source, thereby fostering trust in the generated code.

6. Discussion

Companies using generative AI to create robot code need a reliable, effective way to ensure the responses they receive are traceable and trustworthy. Moreover, there is a growing need for data protection and privacy for IP-protected information, such as proprietary robot code. The approach presented in the paper demonstrates that RAG-based architectures can ensure reliability, transparency, and domain adaptation for AI-powered code generation compared to regular LLMs such as ChatGPT. By grounding code generation in specifically curated, IP-protected data sources, the proposed RAG system addresses major concerns for companies, such as hallucinations, source explainability, and data privacy, especially for safety and process-critical operations such as robotics.

However, the current proposal is limited by the use of a fixed top-k retrieval strategy, i.e., retrieving the top-k most relevant chunks of information from the knowledge database. Future work will focus on developing adaptive retrieval mechanisms and multi-model integration to overcome this limitation. These proposed improvements will handle complex, ambiguous queries that combine visual and test inputs. Such an approach can help answer questions that require interpreting text and images, handle ambiguous and noisy queries, and work across diverse domains and data types. In the context of robotics, such an approach can lead to the use of other programming languages besides RAPID, enabling quick transfer of robot code between RAPID and other languages.

This paper highlights that RAG-based code generation enables reliable and trustworthy use of GenAI in manufacturing. Continued development of such systems will help with realising the full potential of AI-powered code generation for robot programming whilst ensuring IP compliance and user trust.

7. Conclusions

This paper demonstrates the use of Retrieval-Augmented Generation (RAG) for reliable, transparent and domain-specific adaptation of generative AI. By grounding robot code generation in IP-protected data sources and enabling a transparent code-generation pipeline, the proposed RAG system addresses critical challenges in the use of generative AI, such as hallucination, source explainability, and data privacy.

The evaluation of the RAG system shows more accurate, context-aware answers than general LLMs, with full traceability from input to output. While there are limitations related to fixed top-k retrieval and a focus on text-based sources, the proposed architecture provides a solid foundation for future enhancements, incorporating adaptive retrieval strategies and multimodal data integration.

Acknowledgements

This research was funded by the Swedish innovation agency, VINNOVA, under grant number 2024-03234. We sincerely thank VINNOVA and all partner companies involved in Project Code Agents: AI-powered end-to-end solutions for flexible manufacturing, for their tremendous support and valuable contributions.

References

[1] Wolber J, Muyang L, Koch D, et al. Large Language Models for Robotics: A Systematic Literature Review on Prompt Engineering. In: Min J, Zhang W, Fleischer J, et al. (eds) *Sustainable Manufacturing Innovations: Focus on New Energy Vehicles, Production Robots, and Software-Defined Manufacturing*. Cham: Springer Nature Switzerland, pp. 201–213.
[2] Qi Z, Jing X. Advances in Large Language Models for Robotics. In: *2024 7th International Conference on Mechatronics, Robotics and Automation (ICMRA)*. Wuhan, China: IEEE, pp. 72–76.
[3] Liu Y, Hu L, Dong X, et al. Benchmarking Retrieval-Augmented Generation: Challenging Domain-specific Datasets Generation Method. In: *2025 International Joint Conference on Neural Networks (IJCNN)*. Rome, Italy: IEEE, pp. 1–8.
[4] Mao K, Liu Z, Qian H, et al. RAG-Studio: Towards In-Domain Adaptation of Retrieval Augmented Generation Through Self-Alignment. In: *Findings of the Association for Computational Linguistics: EMNLP 2024*. Miami, Florida, USA: Association for Computational Linguistics, pp. 725–735.
[5] Bauer A, Wollherr D, Buss M. Human–robot collaboration: a survey. *Int J Humanoid Robot* 2008; 05: 47–66.
[6] Palla D, Slaby A. Evaluation of Generative AI Models in Python Code Generation: A Comparative Study. *IEEE Access* 2025; 13: 65334–65347.
[7] Macaluso A, Cote N, Chitta S. Toward Automated Programming for Robotic Assembly Using ChatGPT. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 17687–17693.
[8] Ren L, Zhou N, Liu G-P, et al. Automatic Generation of PLC Code Based on Finetuned Large Language Models. In: *2025 IEEE 26th China Conference on System Simulation Technology and its Applications (CCSSTA)*, pp. 546–550.
[9] Clever D, Tan R, Stuhlenmiller F, et al. GenAI for Robot System Engineering. In: *2025 IEEE 21st International Conference on Automation Science and Engineering (CASE)*, pp. 1981–1986.
[10] Zhang Z, Wang C, Wang Y, et al. LLM Hallucinations in Practical Code Generation: Phenomena, Mechanism, and Mitigation. *Proc ACM Softw Eng* 2025; 2: ISSTA022:481-ISSTA022:503.
[11] Zhou J, Su X, Fu W, et al. Enhancing intention prediction and interpretability in service robots with LLM and KG. *Sci Rep* 2024; 14: 26999.
[12] Antero U, Blanco F, Oñativia J, et al. Harnessing the Power of Large Language Models for Automated Code Generation and Verification. *Robotics* 2024; 13: 137.
[13] Dakhel AM, Nikanjam A, Khomh F, et al. Generative AI for Software Development: A Family of Studies on Code Generation. In: Nguyen-Duc A, Abrahamsson P, Khomh F (eds) *Generative AI for Effective Software Development*. Cham: Springer Nature Switzerland, pp. 151–172.
[14] Miao X, Oliaro G, Zhang Z, et al. Towards Efficient Generative Large Language Model Serving: A Survey from Algorithms to Systems. *ACM Comput Surv* 2026; 58: 1–37.

[15] Wang S, Zhu T, Liu B, et al. Unique Security and Privacy Threats of Large Language Models: A Comprehensive Survey. *ACM Comput Surv* 2025; 58: 83:1-83:36.
[16] Yang Z, Raman SS, Shah A, et al. Plug in the Safety Chip: Enforcing Constraints for LLM-driven Robot Agents. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 14435–14442.
[17] Silva LMV da, Köcher A, König N, et al. Capability-Driven Skill Generation with LLMs: A RAG-Based Approach for Reusing Existing Libraries and Interfaces. Epub ahead of print 6 May 2025. DOI: 10.48550/arXiv.2505.03295.
[18] Luo H, Wu J, Liu J, et al. Large language model-based code generation for the control of construction assembly robots: A hierarchical generation approach. *Dev Built Environ* 2024; 19: 100488.
[19] Peffers K, Tuunanen T, Rothenberger MA, et al. A Design Science Research Methodology for Information Systems Research. *J Manag Inf Syst* 2007; 24: 45–77.
[20] Hevner AR, March ST, Park J, et al. Design Science in Information Systems Research. *MIS Q* 2004; 28: 75–105.
[21] Lin XV, Mihaylov T, Artetxe M, et al. Few-shot Learning with Multilingual Language Models. Epub ahead of print 10 November 2022. DOI: 10.48550/arXiv.2112.10668.
[22] Chen W, Chen J, Zou F, et al. Vector and line quantization for billion-scale similarity search on GPUs. *Future Gener Comput Syst* 2019; 99: 295–307.
[23] ZHU Y, MA R, ZHENG B, et al. GTS: GPU-based Tree Index for Fast Similarity Search. *Proc ACM Manag Data* 2024; 2: 1–27.
[24] Wang J, Huang JX, Tu X, et al. Utilizing BERT for Information Retrieval: Survey, Applications, Resources, and Challenges. *ACM Comput Surv*; 56. Epub ahead of print 2024. DOI: 10.1145/3648471.
[25] Hambarde KA, Proença H. Information Retrieval: Recent Advances and Beyond. *IEEE Access* 2023; 11: 76581–76604.
[26] Anh-Hoang D, Tran V, Nguyen L-M. Survey and analysis of hallucinations in large language models: attribution to prompting strategies or model behavior. *Front Artif Intell*; 8. Epub ahead of print 30 September 2025. DOI: 10.3389/frai.2025.1622292.

Annex A

Generated RAPID code:

```
RAPID MODULE PickAndplace
Define tool and work object
CONST tooldata Gripper :- [TRUE, [[0,0,100],[1,0,0,0]], [0.5,[0,0,50],
[1,0,0,0],0,0,0]|;
CONST wobjdata Workobject :- [FALSE, TRUE,**, [[0,0,0][1,0,0,0]],
[[0,0,0],[1,0,0,0])];
Define positions
VAR robtarget Point :- [[500, 0, 200], [1,0,0,0] [0,0,0,0)];
VAR robtarget Point :- [[700, 0, 200], [1,0,0,0], [0,0,0,0]];
VAR robtarget safeposition := [[600, 0, 400], [1,0,0,0], [0,0,0,0]);
Main routine
PROC main()
Activate the tool
SetTool (Gripper);
Setworkobject (Workobject);
! Move to a safe position
Move (SafePosition, v100, z50, Gripper);
Move to Point A to pick up the workpiece Movel (PointA, v50, 25, Gripper);
Gripworkpiece();
Move back to the safe position
```

ChatGPT 40

```
MODULE MoveToWeld
! Persistent data for tool, work object, and positions
```

```
PERS tooldata weldTool := [TRUE, [[0,0,250],[1,0,0,0]], [5,[0,0, 100],[1,0,0,0],0,0,0]];
PERS wobjdata weldWobj:= [FALSE, TRUE, '', [0,0,01,[1,0,0,0]], [10,0,01,[1,0,0,0]]];
! Define home and welding positions (robtarget)
CONST robtarget pHome := [[0,0,500),[1,0,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
CONST robtarget pWeld := [[1000,0,800] [1,0,0,0] [0,0,0,0],[99,99,99,99,99,991];
PROC MoveHomeToWeld()
! Move to home position (ensure safe start)
Moved pHome, v1000, fine, weld Tool WObj:=weldWobj;
! Optional: Wait for external signal or safety check here ! WaitDI diSafeReady, 1;
! Move to welding position (linear move for accuracy) MoveL pWeld, v500, fine, weldTool
WObj:=weldWobj;
! Optional: Call welding routine or set output to start weld ! SetDO doWeldStart, 1;
ENDPROC ENDMODULE
```