Robotics and Computer–Integrated Manufacturing 98 (2026) 103154

Contents lists available at ScienceDirect

Robotics and Computer-Integrated Manufacturing

journal homepage: www.elsevier.com/locate/rcim

Full length article

Human–robot collaborative visual inspection with Large Language Models☆

Osama Tasneem, Roel Pieters *

Unit of Automation Technology and Mechanical Engineering, Tampere University, Korkeakoulunkatu 6, Tampere, Finland

ARTICLE INFO

Keywords:
Human–robot collaboration
Natural language processing
Large Language Models
Local LLM
Generative AI
Prompt engineering
Auto path planning
Visual inspection

ABSTRACT

Human–Robot Collaboration (HRC) is gaining traction in advanced manufacturing as industries shift from isolated robotic systems to more collaborative environments. This transition is supported by advancements in automation and more recently, Generative AI. Large Language Models (LLMs) offer new possibilities for intuitive human–robot interaction through natural language. However, the use of natural language as a means remains very limited due to the ambiguous natural language, environmental noise, pronunciation variability, and multiple phrasing styles. Furthermore, cloud-based deployment of LLMs raises concerns about ergonomics and data privacy, especially for industries and countries governed by strict regulatory requirements. To address these challenges, we present a fully offline, closed-loop robotic assistant for visual inspection tasks in HRC settings. The system supports speech-based interaction, where user instructions are transcribed via a Speech-to-Text (STT) model and processed by a locally deployed, code-generating LLM. Guided by a structured prompt, the LLM produces custom responses for robot perception and manipulation. Inspection paths are generated relative to spatial axes or in specific directions and executed with real-time feedback through a Text-to-Speech (TTS) interface, allowing for a much closer interaction with the robot assistant. The system applies a hybrid control method, where the higher-level instructions are generated by LLM along with a perception pipeline, and the lower-level robot control is managed by ROS for safety and reliability. The system is evaluated across a range of experiments, including local LLM comparisons, prompt engineering effectiveness, and inspection performance in both simulated and real-world industrial use cases. Results demonstrate the system's capability to handle complex inspection tasks on objects with varied sizes and geometries, confirming its practicality and robustness in realistic deployment settings. Code and videos are open-source available at: *https://github.com/CuriousLad1000/RoboSpection*.

1. Introduction

Human–Robot Collaboration (HRC) is an established paradigm that is gaining traction in advanced manufacturing sectors, as in 2023, collaborative robots represented about 10.5% of all industrial robots installed globally [1]. These numbers testify the industrial landscape shift towards Industry 5.0 and HRC is expected to play a vital role in enabling more intelligent, adaptive, and human-centric manufacturing systems [2]. HRC generally refers to scenarios in which humans and robots share a common workspace and operate in close physical proximity, where the degree of collaboration is task-dependent. Unlike traditional non-collaborative environments, HRC combines the strengths of both entities: humans provide cognitive flexibility, adaptability, and decision-making capabilities, while robots perform repetitive, physically demanding, or hazardous tasks. This collaboration enhances the flexibility, efficiency, and safety of industrial operations.

Collaborative robots, or cobots, are currently marketed as an easy to program robot system that uses teach pendant and graphical user interfaces (GUI) for programming [1]. This is true for most of the cobots available in the market, however, some studies have revealed that real software programming can still be challenging. Researchers have found that even modest assembly tasks can require complex code and calibration, [3,4] which will require trained operators. Furthermore, if the tasks demand frequent reprogramming, this will incur additional cost and resources to a company and may not be economically feasible. This highlights the need to find a more suitable alternative than GUI-based programming for better ergonomics and close human–robot collaboration.

Natural language presents an intuitive interface for robot control, offering a familiar mode of interaction for human operators. However, natural language remains inherently ambiguous, context-dependent, and variable across dialects and speakers. Speech-based interfaces, in

☆ This article is part of a Special issue entitled: 'AGI4RoboticsManufacturing' published in Robotics and Computer-Integrated Manufacturing.

* Corresponding author.
E-mail address: roel.pieters@tuni.fi (R. Pieters).

https://doi.org/10.1016/j.rcim.2025.103154
Received 29 June 2025; Received in revised form 22 August 2025; Accepted 25 September 2025
Available online 7 October 2025
0736-5845/© 2025 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

particular, face additional challenges due to noise and variations due to pronunciation and phrasing. Advancements in generative AI and LLMs demonstrate a remarkable ability to understand and retain contextual information that earlier NLP systems lacked [5]. These models can maintain a limited memory of preceding dialogues, enabling them to generate contextually appropriate responses over the course of a conversation. When integrated into robotic systems, such capabilities can significantly enhance human–robot interaction. Users will be able to concentrate on higher-level planning and decision-making while offloading the intricate details of a task to the robot system. This not only reduces the cognitive load on the user but also facilitates more natural and efficient collaboration with the robotic system.

However, almost all of the existing LLM-based solutions present challenges when it comes to deploying such systems in industries. LLMs are generalist models that hold large amounts of generalized information. These models are very resource intensive and require a lot of time, money and compute to train. Even after training, such models can require quite large investment to locally deploy. Most of the present solutions utilize cloud-based services, that require users to send data to the cloud for processing. This raises another concern regarding data security and privacy for the industries, and the operators using the system. This practice may not comply with company policies and may become a hindrance in utilizing such technology on the shop floors. To provide a solution to the mentioned challenges we propose an extension to our previous research work [6], which presented an open-source intuitive GUI-based tool that assists an operator in visual inspection tasks with the help of a cobot manipulator by automating the generation of inspection paths.

This work presents an offline, closed-loop system for HRC in visual inspection tasks, using speech-driven interfaces and locally deployed LLMs. The system enables natural communication by allowing operators to issue inspection commands via speech. These commands are transcribed using a Speech-to-Text (STT) model and passed to a locally hosted code-generating LLM. The LLM, guided by a structured system prompt, generates an appropriate response tailored for robotic manipulators and perception-related tasks. The generated inspection paths can be defined relative to spatial axes or in a specific direction based on the user's intent. The generated code is either previewed or executed directly on the robot, with real-time feedback delivered via a Text-to-Speech (TTS) system, completing a bi-directional communication loop. Additionally, the system uses a hybrid control method, where higher-level instructions are generated by the LLM and lower-level control is managed by ROS for safety. The system is evaluated through experiments comparing local LLMs for code generation, prompt engineering strategies, and their effectiveness in autonomous visual inspection scenarios. This approach supports privacy-preserving deployment and flexible inspection path generation. Additionally, the robot assistant is tested on real industrial use case in simulated and real environments with varying complexities.

The main contributions of this work are:

- A system that enables natural language interaction for human–robot collaboration in visual inspection tasks
- Locally hosted LLM that generates task-specific robot code from speech, to customize inspection paths
- Hybrid control framework that combines high-level LLM planning with low-level ROS execution ensuring flexible and safe robot behavior
- Validation of the system using real-life industrial use case in both simulated and real world environments, across varying levels of complexity

2. Related work

This section provides a detailed overview of related work done in this field along with comparison to our current work.

2.1. HRC in Industry 5.0

The transition from Industry 4.0 to Industry 5.0 brings the collaboration between humans and robots to the center stage. According to the International Federation of Robotics (IFR), around 541,302 robots were newly installed in factories in 2022. Approximately 50 to 60 thousand of these are collaborative robots (cobots) [1]. Several case studies by IFR demonstrate the use of cobots for quality assurance tests in collaboration with a human operator [7] and the collaborative assembly of lights [8]. The rise of newer technologies also brings about various challenges to the shop floor of industries, including the retraining of the operators, making them accustomed to a newer work environment in close proximity of cobots. In addition, occupational health and safety criteria are vital in the implementation of collaborative robotics [9]. Workers worldwide are exposed to occupational hazards. An ergonomic intervention using HRC-based solutions may help alleviate some of those risks. An overview presented by Marta et al. highlights the existing ergonomics assessment tools as well as the available monitoring technologies to drive and adapt a cobot's behavior [10]. Having good cognitive ergonomics is essential for efficient and reliable human robot collaboration. The interaction between the robot and the human must be designed to take full advantage of human intelligence and the system must be intuitive enough to reduce the cognitive load on the human operator. Despite the emergence of various multimodal interactions, the off-line programming using graphical user interface (GUI) remains the most widely used means of interaction between humans and robot [11] in industrial settings.

Visual inspection **-** Inspection-related tasks are crucial for manufacturing industries and often performed in the form of quality checks on products leaving a production line to ensure products reach the consumer free from any defects. Visual inspection is also performed on industrial machines as a part of maintenance procedures and often requires a special technician that manually inspects certain parts of a machine to ensure optimal performance. Such inspection tasks are costly due to the downtime and the labor involved. Furthermore, this may expose the human inspector to hazardous environment. According to the estimates provided by the European Agency for Safety and Health at Work (EU-OSHA), approximately three out of five workers suffer from Musculoskeletal Disorder (MSD), among which backache and upper limb pain are the most common [10,12]. A potential solution may be to automate the inspection task with the help of a robot.

One notable approach is presented by Lončarević et al. [13], where the authors proposed a method for generating inspection trajectories using the 3D CAD model of the target object. While effective, this approach assumes the availability of the CAD model, which may not be possible in many real-world inspection scenarios involving unknown or unmodeled objects. Hence, alternative methods are required to allow inspection of such objects without relying on pre-existing CAD data.

Our previous research [6] attempted to mitigate these challenges by developing an open-source, GUI-based tool that, in conjunction with a collaborative robot, assists a human inspector in performing visual inspection tasks through automated generation of inspection paths. The proposed solution was both robot and object-agnostic, capable of operating with a wide range of objects irrespective of their size or shape.

2.2. Existing HRC frameworks

Learning by Demonstration for HRC Tasks **-** Learning by Demonstration (LbD) is a research paradigm that allows non-expert users to teach robots new skills by having them observe human demonstrations. In a survey by Mukherjee et al. [14], the state of the art and emerging trends in LbD were reviewed. The work presented by Cakmak M. and Thomaz Andrea [15] emphasized the importance of a bi-directional learning process between humans and robots. The authors identified three types of queries, and their experiments showed that

robots performed best when using feature-based queries, particularly those posed as closed-form questions (e.g., asking directly for positions and orientations), which effectively constrained the range of possible answers. However, limiting to single answers like yes/no results in inaccuracies or missed information for the system. In addition, the absence of memory and contextual awareness limits the system's ability to function as a reliable assistant.

Reinforcement Learning based approach - Reinforcement Learning (RL) is a paradigm in which robots learn sequential decision-making by interacting with their environment, with the goal of maximizing a reward function over time. Koch et al. [16] applied RL to address the View Planning Problem, allowing the agent to generate state–action sequences that produced optimal viewpoints for object inspection. Their approach achieved up to 90% coverage with only a few poses. However, the use of RL would require a very large dataset along with longer training times to train the system for a particular task due to sparsity of rewards in the real-world scenario. In addition, Zhang et al. [17], in a recent survey, emphasized that deep learning–based approaches introduce additional challenges for intelligent manufacturing, including poor generalization capability due to inadequate representation of unstructured data, weak integration of domain knowledge, and a scarcity of high-quality training datasets. In contrast, our system does not require any training data for inspection tasks and uses pre-trained models for code generation. Additionally, the absence of human–robot interaction limits the RL based system's adaptability in dynamic scenarios and reduces its applicability in tasks beyond visual inspection without retraining.

2.3. Natural language interfaces for robotics

The level of programming skills required for robotics grows along with the increase in task complexity. In recent years, researchers have been investigating the use of natural language as a more intuitive modality for human–robot interaction [18], with the aim of replacing conventional GUI-based interfaces. Previous research has demonstrated grounding language by mapping it to the robot control commands [19] or by providing environmental context and task constraints along with grounding language [20]. Kollar et al. [21] proposed a system that interprets natural language instructions by extracting a sequence of spatial description clauses from the input and inferring the most likely path through the environment. Thomason et al. [22] introduces a dialog agent that utilized semantic parsing to comprehend human instructions. The system engages in human–robot conversations to incrementally learn and resolve ambiguities using a dialog manager. However, research done in this domain remains limited due to several challenges that arise due to the inherent nature of natural language. Natural language is often vague or context-dependent, can change from person to person and speech recognition and interpretation systems can introduce considerable delay in the interaction.

2.4. Advances in generative AI and LLMs

Recent advances in generative Artificial Intelligence have led to further developments in Natural Language Processing (NLP). The emergence of the Transformer architecture [23] has led to rapid growth in the development of highly capable Large Language Models (LLMs). For example, LLMs can be fine-tuned on extensive code repositories combined with natural language across multiple programming languages [24–26]. This training enables LLMs to understand programming syntax and effectively generate code in various programming languages based on natural language input. Such capabilities hold significant potential for the generalization ability [17] across various domains such as the automated generation of robot control APIs. The work presented by Vemprala et al. [27] presented an experimental study on using prompts efficiently with OpenAI's ChatGPT to control a robot. Arenas et al. [28] presented various prompting paradigms for efficient code generation. Additional development in the field has led to the emergence of neural speech recognition models that convert audio to transcription commonly known as Speech To Text (STT), such as Whisper [29]. These models are trained on large corpus of text and user voice in multiple languages with different dialects. This makes the model a highly capable language transcriber and translator.

2.5. Integration of genAI with robotics for HRC

The task knowledge gained by robotic systems with the help of LLMs is only meaningful if the system can translate it into appropriate physical actions. Researchers are currently exploring two prominent approaches to enable robots to act effectively on contextual information. Robotics inherently integrates technologies from multiple domains, with variations in robot types, degrees of freedom, and kinematic structures. At present, the low-level control in robotics relies on deterministic mathematical models, that enables precise and reliable motion planning. However, when such control is passed on to an AI system, especially generative models, the reliability is reduced due to their black-box nature. The output of these models is heavily dependent on the quality and diversity of training data. This makes the "End-to-End" generative AI robot control non-deterministic and comparatively less reliable. Unlike domains such as text or image processing, where large datasets are readily available, robotic systems require highly specific data, including joint values, torque feedback, and multi-modal sensor inputs, to enable robust end-to-end control. Currently, the scarcity of such datasets presents a significant challenge to achieve accuracy levels above 95% for industrial tasks. Despite these challenges, there have been many efforts by various research groups to try and train an end-to-end model with limited amount of data available.

Brohan et al. [30,31] presented a task-agnostic Transformer model for robots. The model demonstrated good generalization capabilities across different environment settings. The authors highlighted the importance of diversity in data as compared to the data size for better generalization and performance. However, a key limitation identified is that the physical capabilities of a robot remain constrained by the distribution of skills represented in the training data, limiting its ability to perform tasks beyond those encountered during training. The robots particularly failed at tasks related to grasping objects by specific parts and precision motion. The work presented by Kim et al. [32], developed a model that used a fused vision encoder that maps vision inputs to image patched embeddings, a projector converts embeddings to be used by LLMs and then the LLM backbone is used to predict next action token which can be fed to robot after decoding for direct low level control. This model also experiences similar limitations related to accuracy, speed, and the use of single-image observations to generate outputs.

Another direction of research involves splitting the cognitive and control into separate parts, this approach utilizes the hybrid of generative AI for higher level tasks planning and traditional approaches for lower level safety and control. Generative AI can be used for cognitive decision making tasks, providing assistance to user, generating path plans and generating robot specific code and APIs for robot control. Whereas, the lower level control is still governed by traditional mathematical approaches to maintain safety, accuracy and reliability. This approach benefits from not requiring robotic data instead, the only data required at that time can be fetched through various onboard sensors such as a depth camera to perceive environment.

Liang et al. [33] demonstrated that code-generating LLMs can be repurposed to generate robot policy code with natural language as input. The authors observed a degree of generalization based on LLM used without requiring a huge amount of data, in contrast to end-to-end learning. However, the capabilities are heavily dependent on the LLM being used. The authors also stated another limitation pertaining to the inability of interpreting larger and more complex commands.

The solution presented by Liu et al. [34], utilized a prompted GPT-4 LLM to split the high-level language commands into sequence

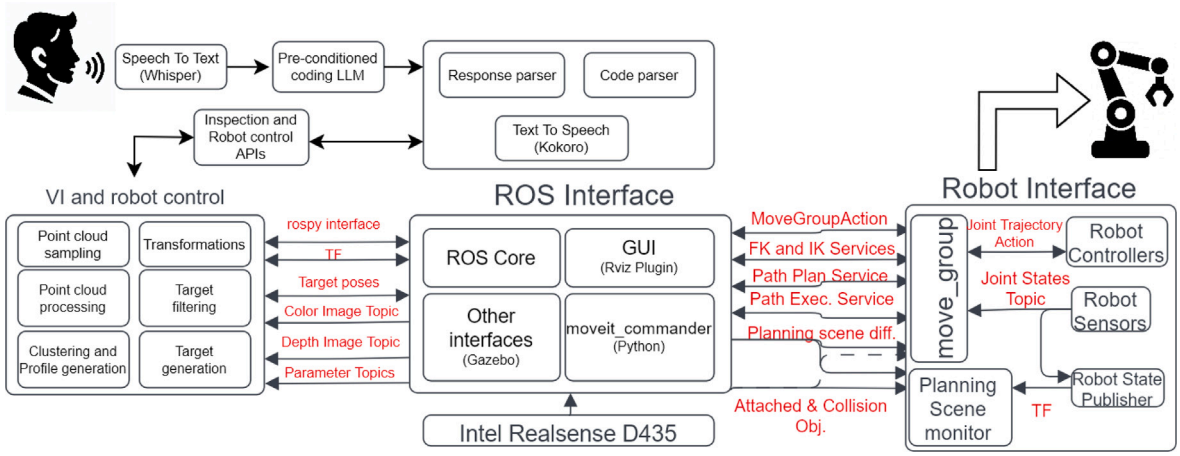


Fig. 1. Block diagram depicting the flow of information between various modules.

of motions to be executed by the robot. The authors implemented a feedback system where, if the robot fails to perform an action, the action is performed by the user via tele-operation and saved by the system for future use. The system performed well on short-horizon tasks despite some failures due to task complexity. However, the system failed on long-horizon tasks due to the accumulation of errors.

2.6. Comparison to our approach

Most of the present solutions working in industries are focused on GUI based control. Our research focuses on an approach where the natural language is used as a modality for bidirectional conversation with an HRC based system to assist in visual inspection tasks. For control, our system utilizes a hybrid approach instead of an end-to-end model and hence, does not require any robot data for training. In our approach, the robot's physical motions are guided by the structure of the object under inspection and so, the robot's motions are only limited by the kinematic constraints and not the training data. Our system is designed to handle longer and more complex user instructions, offering step-by-step assistance without requiring the operator to recall task procedures. This not only reduces the cognitive burden on the user, but also enhances ergonomics and produces a more intuitive and human-centric human–robot interaction. The entire system is locally deployable and requires minimal computational resources, which promotes data security and privacy. This makes it particularly well-suited for industrial environments where data sensitivity is a concern and reliance on cloud-based processing is undesirable.

3. Methods

The objective is to present the development of a closed-loop, HRC-based visual inspection framework that enables speech-driven interaction between the user and the robot assistant and is suitable for local industrial deployment. The system uses a locally deployed LLM to support users in generating path plans, guiding robot motion around objects, and automating key aspects of the visual inspection process. This section provides a detailed overview of the proposed system. Fig. 1 depicts a block diagram of the overall information flow between the various modules used in the project.

The user interacts with the system through spoken instructions, which are converted into text. This text is passed on to a Pre-conditioned coding LLM that generates response in context to the provided instruction. The LLM output is forwarded to another module, where it is parsed and used to invoke the relevant inspection and robot control APIs from the codebase. At a lower level, these APIs interface with the Visual Inspection (VI) and robot control modules (left block), which handles core inspection tasks such as point cloud sampling, processing, clustering, target filtering, and target generation.

The generated targets are sent to the ROS interface that uses a publish–subscribe architecture to establish communication between interconnected nodes. Physical devices such as the depth camera and the Robot's controller are directly connected to the ROS interface that provides the low-level robot control in real-time. Core services include TransFormation (TF) publishing under the TF tree, forward (FK) and inverse kinematics (IK), the MoveGroup action service, path planning, and the planning scene interface. These services interact with the robot through MoveGroup, while the robot controller passes critical sensor and actuator data to maintain real-time operation. Once an action is completed, the result is transmitted to the user through a text-to-speech module, giving the robot a voice.

3.1. Neural speech processing

The interaction between the human and the robotic system requires a shift in modality from audio to text in order to interpret the user's intent. This can be achieved by using a neural speech processing model that is pre-trained to convert speech into text such as Whisper [29]. Whisper is an encoder–decoder based Automatic Speech Recognition (ASR) model that is pre-trained on 680,000 h of labeled speech data annotated using large-scale weak supervision. The model generalizes well to various domains without needing any special fine-tuning. Whisper models are developed in five different configurations, each corresponding to a different model size. Generally, larger models trained with higher number of parameters using larger datasets produce higher accuracy. However, given our objective of deploying the system with minimal computational resources, we used the whisper-small-en variant, with 244 million parameters, which offered satisfactory performance in our case.

3.2. LLMs for code generation

The LLMs are generally trained on large corpus of text that provides the models with ability to generalize and produce response to user's query in a requested form. Fine-tuning such models on large amounts of source code allows the model to generalize and generate code while following the given constraints. The system uses a small and highly efficient coding LLM that takes the input from ASR model in the form of text and generates response based on the instructions.

Additionally, In order to have a seamless HRC experience while keeping the resource requirements to minimum, we created and performed a benchmark evaluation against multiple coding and non coding-based generative models that were specifically pre-conditioned to cater our requirements as discussed in Section 5. We used Qwen2.5-Coder-1.5B [25,35] for code generation. The model offered significant improvements in code generation, code reasoning, and code fixing related tasks while keeping the model size to minimum.

3.2.1. Pre-conditioning the LLM

Depending on the nature of the task, the system prompt can be used as contextual input to supply the LLM with important task-related information. In practice, the system prompt acts as a structured guide or constraint, defining the rules, available APIs, and operational boundaries that the model must follow when generating responses. Since LLMs generate output based on patterns learned during training, without grounding, they may produce irrelevant, incomplete, or hallucinated results. By embedding task-specific details, such as robot control APIs, task constraints, and expected response formats, into the system prompt, we ensure that the LLM's output remains relevant, accurate, and aligned with the capabilities of the inspection framework. The System prompt presented here is only a small portion of the complete version, which was shortened to maintain brevity. The complete system prompt can be accessed in the project's GitHub repository4.

To provide a seamless HRC experience for our visual inspection use case, the coding model was pre-conditioned using a task-specific system prompt, allowing the LLM to concentrate on tasks relevant to visual inspection. Here, in the trimmed version of system prompt, the text in red is the string that is provided to the coding model as a context which details the nature of the task along with the process details and defining the objective for the robot assistant. The system is also provided with some examples in a few-shot manner, which mitigates hallucinations and generates a grounded response.

```
# System prompt
"Imagine we are working on HRC based path planning system using a
↪ manipulator robot. The robotic arm has a depth camera attached to
↪ its end effector. I would like you to assist me in interacting
↪ with the system and sending commands to this robot.
There are five main steps that are required to complete the
↪ Inspection.
STEP 1: Fetch point cloud
STEP 2: Cluster point cloud
STEP 3: Generate Inspection path
STEP 4: Create robot targets
STEP 5: Run through targets OR plan and execute path
If the user asks what to do, briefly explain in few lines these steps
↪  on higher level without giving function names or generating any
↪ code. Do not tell how to call functions. Instead, tell the user
↪ simple commands that can be called to complete the steps."

# For example:
"Me: Who are you?
You: I am an AI assistant programmed to assist you with HRC based
↪ path planning system using a manipulator robot."
```

The system prompt does not restrict the code generation but instead prevents the entire code from leaking. For example, if a user asks for available features, then the robot assistant responds by informing the user of available processes and commands that can be used to complete a task instead of directly outputting lines of code. This design choice was necessary because the LLM used in this work is fine-tuned to generate code rather than simple text.

In addition, the robot is provided with the knowledge of Visual Inspection and low-level robot control APIs which the robot assistant can use to complete a task. An example of such an API, added within the system prompt, is shown below. Each API is presented in a defined format together with its description of use.

```
# Example API:
#This function, when called, selects the centered profile around the
↪  y axis.
<code>
thread_handle.select_centered_profile_around_y()
</code>
```

This ensures that the codes generated by the assistant are grounded and follow the correct sequence.

3.2.2. Code processing

The reliability and executability of the generated code are crucial to ensure that the robot assistant can successfully complete the task. In Section 3.2.1 we highlighted the importance of grounding the model's response with the help of a well-structured prompt to ensure reliable code generation. The generated code is enclosed withing XML tags <code>< /code> which makes it easier for post-processing operations. The generated code is parsed through the parsing module to extract the code and the text reply from assistant to user. This module verifies the correctness of the code before execution. If the code parser detects any errors, the system informs the operator by telling him that there was an error in executing the command and the generated code is discarded without any execution. On the other hand, the robot assistant also informs the operator of the successful execution if the task is completed successfully.

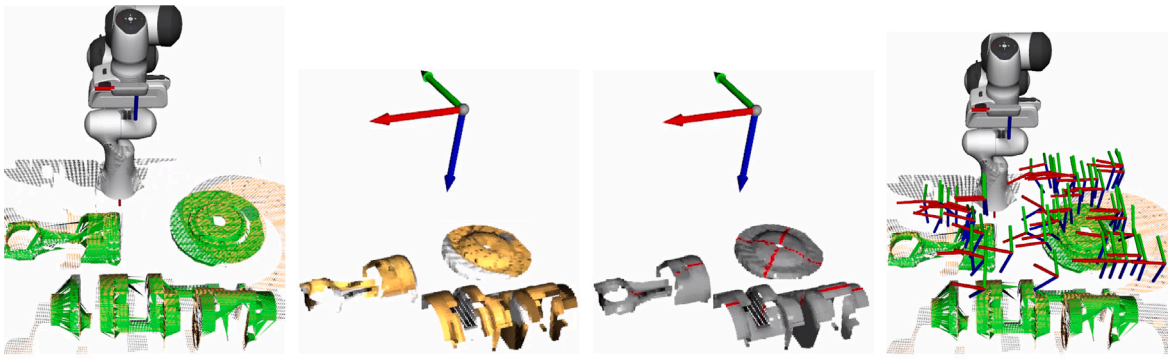
The code is forwarded to an execution module while the textual reply is sent to a text-to-speech (TTS) model that converts the text into audio form to convey the message from the assistant to the user. For this we use kokoro [36,37], which despite its compact architecture with 82 million parameters, achieves speech synthesis quality comparable to much larger models, while offering faster speed and lower resource consumption.

3.3. Collision avoidance in complex environment

Collision avoidance is critical when inspecting complex objects, especially in cluttered scenes involving multiple objects. In our previous work [6], we implemented target filtering and removal strategies to eliminate hazardous targets and mitigate the risk of collision. However, the motion planner occasionally still produced trajectories that resulted in collisions with nearby objects. In this work, we introduced a new collision avoidance pipeline that utilized the same depth camera mounted on the end-effector to generate an object-aware scene. This scene is then provided to the low-level motion planner [38], enabling it to plan paths that avoid contact with any objects. The depth camera captures, filters, and converts the scene into a mesh, which is then used for collision-free motion planning.

3.4. Multi-object inspection

The system is capable of recognizing and inspecting multiple objects at the same time. If multiple objects are placed within the field-of-view of the camera and are within the reach of the robot as shown in Fig. 2(a), the robot assistant performs segmentation on the objects and extracts each object separately (Fig. 2(b)). Once extracted, the assistant asks to select inspection profiles individually around each object while giving the user the flexibility of selecting independent path profiles for target generation (Fig. 2(c)). Furthermore, these profiles are directed to further processing and converted to individual targets (Fig. 2(d)). Each target corresponds to a specific camera pose that the camera, mounted on the robot's end-effector, must reach to collect data from that position. The direction and orientation of each pose are determined by the inspection profiles selected by the user with the help of the robot



(a) (b) (c) (d)

Fig. 2. Simulation of Multi-object inspection. (a) Point cloud with collision mesh (green). (b) Object selection interface. (c) Profile generation and preview. (d) Inspection target generation. All coordinate frames depict the camera target pose.

assistant. These poses are sent to low-level robot controls for motion planning, while using the generated collision-aware mesh to provide a collision-free path around each object. Fig. 2 gives an overview of the entire process, while Fig. 5 visualizes movements around individual objects.

4. Visual inspection use case

Visual inspection is an integral part of all manufacturing industries. This research was produced in response to an industrial challenge presented by a paper mill industry. In the paper mill industry, suction rolls are essential to the production process. Each suction roll is approximately 11 meters long and 2 meters in diameter and has approximately 100,000 to 500,000 holes on the surface, depending on the size of the role. These suction rolls are used for transporting the sheets of paper, dewatering the felt, and removing moisture from the pulp during the paper production process. This process leads to the blockage of thousands of holes and requires manual intervention from a human inspector. This leads to additional downtime, increased costs, and ergonomic challenges for the inspector. A promising alternative is to utilize the HRC based robotic system enhanced by LLM, with the aim of supporting human inspectors in performing the task more rapidly and with improved ergonomics.

4.1. Integration

All developments are integrated with ROS[1] nodes and ROS moveit[2] APIs for low-level control. For robot and perception hardware, we utilize the Franka Emika collaborative robot[3] and an [Intel Realsense D435 RGB-D camera. AI and LLM based computations are done on an Ubuntu PC using a single Nvidia RTX 3060 GPU with 12 GB VRAM, running ROS Noetic.] A headset with standard microphone is used as means to listen and speak to the system. Gazebo physics engine along with Rviz is used as the simulation environment for testing with simulated objects. All algorithms are developed and integrated using Python programming language and run within Jupyter notebook. For testing, multiple objects were selected that varied in size and geometry.

4.2. Path planning for visual inspection

Whenever a user provides an instruction to the assistant, the assistant invokes a specific API from the codebase to complete the desired task. A detailed system prompt, listing all APIs used by the assistant, is available in the GitHub repository[4] along with the complete code and resources. All APIs were developed based on our previous work [6] along with some new additions.

We used a single depth camera mounted on the robot's end-effector to aid in the automated inspection task.

The depth camera captures several sample frames of the scene and generates a robust point cloud of the object under inspection after applying various filtering processes to eliminate irrelevant information from the data. The handling and processing of point cloud data is done with the help of Open3D [46]. The point cloud undergoes an additional segmentation stage, providing the user with the flexibility to identify and select specific objects for inspection. In our current implementation, this process is designed to be more collaborative, allowing the user to interact with the system through natural language commands. By issuing contextually meaningful instructions, the user can direct the assistant to perform various operations on the objects. These instructions need not be precise, but should convey the user's intent as is done in natural human-to-human communication. Some of the examples are listed below.

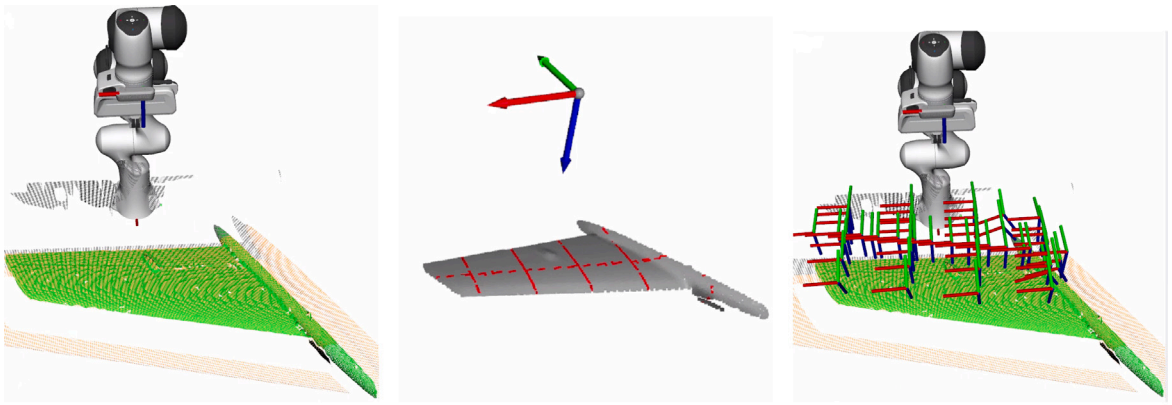
```
Example 1: "Hey Franka! show next object."
Example 2: "Hey Franka! select this object."
Example 3: "Hey Franka! rotate object."
Example 4: "Hey Franka! show previous object."
Example 5: "Hey Franka! reset view."
```

Following this stage, the user may instruct the assistant to generate various inspection paths around the object. The assistant plans and generates various paths around the object collaboratively. The user can provide specific directions as a guide to generate inspection paths.

```
Example 1: "Hey Franka! generate inspection profiles!"
Example 2: "Hey Franka! select centered profile around x axis."
Example 3: "Hey Franka! Please select five profiles around y."
Example 4: "Hey Franka! select profile forty five degrees towards
↪ right of the object."
Example 5: "How many profiles are there around z axis?"
```

The generated profiles are then filtered and converted to actionable robot targets. After which, the user may instruct the robot assistant to either move through each target independently or go through all at once while following a smoother and more consistent motion path. For the entire process, the user need not to remember the order of the process or the instructions for conducting the inspection task and

[1] https://www.ros.org/
[2] https://moveit.picknik.ai/
[3] https://franka.de/
[4] https://github.com/CuriousLad1000/RoboSpection



(a) (b) (c)

Fig. 3. Simulation of Single-object inspection. (a) Point cloud with collision mesh (green). (b) Profile generation and preview. (c) Inspection target generation. All coordinate frames depict the camera target pose.

Table 1
LLM model benchmark results for visual inspection task. Total time indicates the accumulated time for 230 prompts.

Model	Precision	Recall	F1 Score	Average time per prompt (s)	Total time (s)	VRAM used (GB)
Qwen2.5-1.5B-Instruct [39]	1	0.687	0.814	1.37	314.17	3.4
Qwen2.5-Coder-1.5B-Instruct [35]	1	0.757	0.861	1.29	296.48	**3.4**
Qwen2.5-Coder-3B-Instruct [40]	1	0.813	0.897	1.87	430.52	6.37
Qwen2.5-Coder-7B-Instruct (Q8) [41]	1	0.865	0.928	**1.17**	**268.71**	7.71
Deepseek-Coder-7B (Q8) [42]	1	0.435	0.606	0.75	172	8.68
Deepseek-Coder-1.3B-instruct [43]	1	0.490	0.663	4.85	1116.2	4.54
DeepSeek-Coder-V2-Lite-Instruct (Q4_KM) [44]	1	0.778	0.875	2.3	527.89	10.97
Starcoder2-15B (IQ4_XS) [45]	1	**0.887**	**0.940**	5.83	1340.7	8.51

can ask the assistant for help at any stage. Fig. 3 provides a general overview of the process, whereas, Fig. 4(b) depicts the robot motion across the wing of a plane.

```
Example: "Hey Franka! What should I do next?"
```

5. Results and discussion

5.1. Model comparative for visual inspection task

Selecting an appropriate large language model (LLM) is critical for developing an effective natural language-based robotic assistant for human–robot collaboration (HRC). The ideal model should strike an optimal balance among three essential criteria: high accuracy, fast response generation, and minimal computational resource requirements for local deployment. The following experimental setup was designed to identify an LLM suitable for local deployment. A benchmark dataset was prepared containing 230 inspection-related prompts along with their ground truth. Eight candidate models were selected based on their parameter size, fine-tuning status, and quantization level. In general, larger models tend to yield higher accuracy but require significantly more GPU memory and exhibit slower response times. The selected models span a range of sizes, from 1.3 billion to 15 billion parameters. Models specifically fine-tuned for coding and instruction-following tasks demonstrated superior performance in code generation scenarios. The quantization technique is used on larger models to reduce the memory footprint and computational requirements of larger models at the cost of a slight drop in accuracy. Table 1 summarizes the evaluation of eight large language models (LLMs), both coding- and non-coding-oriented, based on four key metrics: accuracy, average generation time per prompt, total evaluation time for all 230 prompts collectively, and VRAM usage. Among the candidates, Qwen2.5-Coder-1.5B-Instruct [35] demonstrated the lowest memory consumption, utilizing just 3.4 GB of VRAM, while maintaining a competitive accuracy of 86% and an average response time of 1.29 s, making it the most balanced model across all criteria. In contrast, the Q4-quantized DeepSeek-Coder-V2-Lite-Instruct [44] recorded the highest VRAM usage, despite producing accurate outputs with 87% task success. The DeepSeek-Coder-7B [42], with 8-bit quantization, delivered the lowest accuracy among all candidates while consuming significantly more memory than most models, suggesting poor efficiency. The Qwen2.5-Coder-7B-Instruct [41] emerged as the fastest model, generating responses in just 1.17 s per prompt, while also achieving an impressive accuracy of 92%, the second highest overall. The Starcoder2-15B [45], though the most accurate with a 94% success rate using 4-bit integer quantization, was the slowest, with an average generation time of 5.8 s per prompt, nearly five times slower than the Qwen2.5-Coder-7B-Instruct. Finally, the general-purpose Qwen2.5-1.5B-Instruct [39] (non-coding model) also showed reasonable performance, using only 3.4 GB of VRAM.

5.2. Prompt adherence and robustness

The natural language-based HRC system must be robust against variations in the natural language instructions given by the user. Furthermore, the LLM being used as the processor must be able to understand the context and adhere to the prompts provided by the user and the system. This is crucial for a seamless HRC interaction between the user and the robot assistant.

To evaluate the prompt adherence and robustness of the system, the following experiment was designed. The visual inspection workflow consists of five core steps necessary to complete the task. In addition, users can issue 18 additional commands to request or provide information to the robot assistant. A dataset of 230 prompts was created, with each instruction represented by 10 variations. The dataset is also annotated with corresponding ground-truth responses for evaluation. The experiment is performed using the Qwen2.5-Coder-1.5B-Instruct [25]

Table 2
Prompt adherence and robustness results for 230 prompts (truncated to 10 for brevity).

Prompt	No. of variations	Feasibility	MESR	Task success rate
fetch point cloud	10	0.8	0.8	80%
cluster point cloud	10	0.2	0.2	20%
generate inspection path	10	0.9	0.9	90%
create robot targets	10	0	0.6	0%
plan and execute path	10	0.2	0.2	20%
exit viewer	10	1	1	100%
show next object	10	1	1	100%
How many profiles are available around _, _ and _ axis?	10	0.6	1	100%
select _ profiles around _ axis	10	1	1	100%
select profile _ degrees towards _ of the object	10	1	1	100%
13 more prompts...	–	–	–	–
Total	**230**	**0.78**	**0.82**	**79.57%**

Table 3
Closed-loop visual inspection times across four distinct objects and code inference time in seconds.

Prompt	Transmission gearbox	Bevel gear set	Engine block	Plane wing	Code inference time
fetch point cloud	9.15	8.13	9.09	8.25	2.153
cluster point cloud	15.25	14.15	17.17	14.25	6.111
generate inspection path	33.16	31.1	33.03	32.25	2.664
create robot targets	11.13	8.23	22.2	9.1	0.991
plan and execute path	13.1	12.17	13.23	13.05	1.831
Total time (s)	**81.79**	**73.78**	**94.72**	**76.9**	**13.75**

LLM model that provided the best balance of speed, accuracy, and resource usage. The results are presented in Table 2.

```
# Original Prompt: fetch point cloud
variations: ["fetch point cloud", "generate point cloud", "retrieve
↪ point cloud", "get point cloud", "acquire point cloud", "load
↪ point cloud", "obtain point cloud", "collect point cloud", "scan
↪ point cloud", "start point cloud"]
GT:
say("fetching pointcloud".)
pointcloud = camera_processor.load_point_cloud(samples, offset_y,
↪ offset_z, manual_offset, spacing, trim_base, Hide_prev=False,
↪ Dbug=dbug, eval_tag=False)
```

Each prompt is fed to the LLM model, and evaluation scores are generated on the basis of:

Feasibility: The generated code respects instructional constraints and avoids hallucinations or illegal constructs. It measures whether the generated code fully matches the expected ground truth, including syntax and all required code lines. This score is quantitatively represented as the ratio of successful generations to the total code generations for a particular task.

Minimum Execution Success Rate (MESR): checks whether the generated code achieves the minimum required functionality, even if minor code segments are missing. The code must execute the core logic without syntax or runtime errors. This score is quantified as the ratio of critical code construct generations with successful executions to the total code generations for a particular task. For example, in Table 2, the instruction "create robot targets" shows a Feasibility score of 0 and MESR score of 0.6, indicating that none of the 10 prompt variations produced a perfect match with the ground truth, but, 6 out of 10 successfully completed the intended task, despite minor omissions (e.g., not informing the operator of task completion).

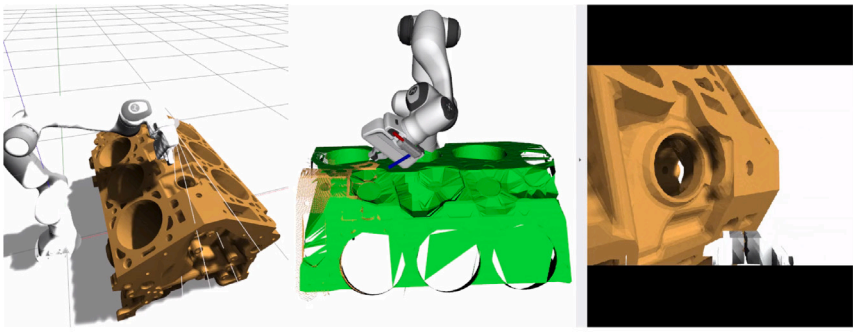
Task Success Rate: checks if the task is completed successfully. Some minor changes are allowed that do not impact the interaction between the human and the robot such as the assistant notifying the user in a rephrased sentence without changing it's semantic meaning or different code is generated that do not affect the functionality or user experience. It is quantified as the percentage of all successful task completions for a given prompt. For example, in Table 2, the instruction "How many profiles are available around_, _ and _?", has 10 variations based on which 10 responses were generated. Although only 6 responses exactly matched the ground truth, all 10 responses successfully answered the query, resulting in a Task Success Rate of 100%. This indicates that the remaining 4 generated responses also successfully completed the task, with the system providing the correct answer to the question, even though possibly in a rephrased sentence form.

The results presented in Table 2 show that the model performs well on average, with a feasibility score of 0.78, a Minimum Execution Success Rate (MESR) of 0.82, and a task success rate of 79.57%. Prompts such as *exit viewer* and those related to profile selection achieved perfect scores, indicating strong adherence to instructions and robustness to natural language variation. These results suggest that the model effectively handles well-defined and narrowly scoped prompts. However, prompts such as *cluster point cloud* and *plan and execute path* showed poor adherence. Furthermore, the *create robot targets* prompt achieved a MESR score of 0.6 despite a feasibility score of 0 indicating that the model successfully generated the critical part of the code, but failed to generate the minor code that may impact the human–robot interaction.

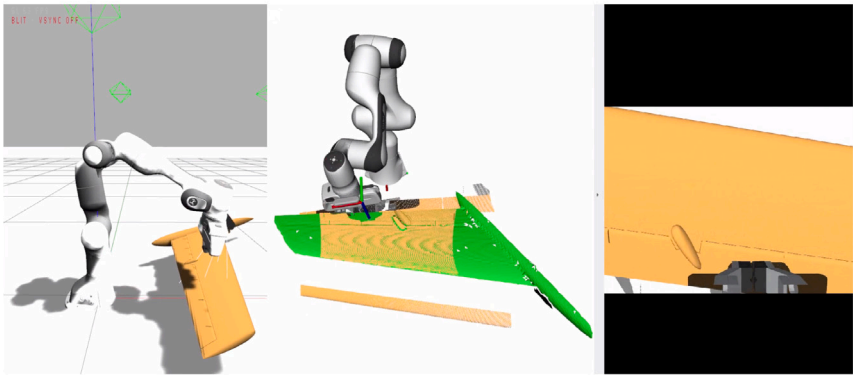
5.3. Closed-loop execution efficiency

To evaluate the closed-loop efficiency and performance of the chosen Qwen2.5-Coder-1.5B-Instruct model [25], we carried out closed-loop experiments on various objects. From a dataset of 23 prompts, we selected 5 key prompts that represent the major steps necessary to complete a visual inspection task. Additionally, to obtain consistent timing measurements independent of the inspected object, we recorded the code inference time for these five instruction prompts.

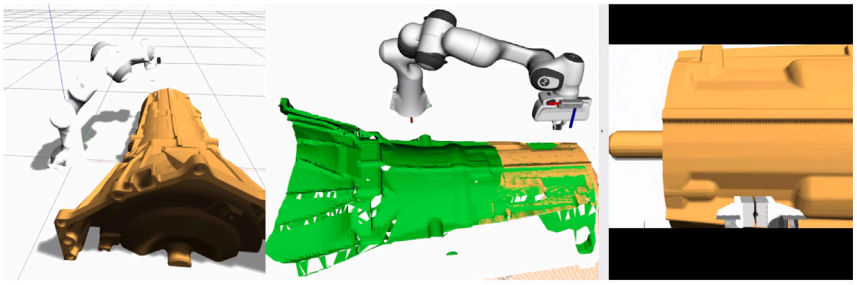
Table 3 presents the recorded execution times for the selected prompts across four different objects in a closed-loop setting. The timings for each prompt include speech-to-text conversion, code generation, code parsing, code execution (with API calls and background processing), and the feedback returned by the robot assistant. Robot motion times are excluded, as they vary significantly depending on the size and geometry of the object, as well as the inspection profiles chosen by the operator during the path planning. Furthermore, Table 3 provides the code inference times, representing the time required by the model to generate code for a given prompt. From these results, it can be observed that the total code inference time during closed-loop execution is significantly less than the overall task execution times for different objects, which is due to the factors discussed earlier.



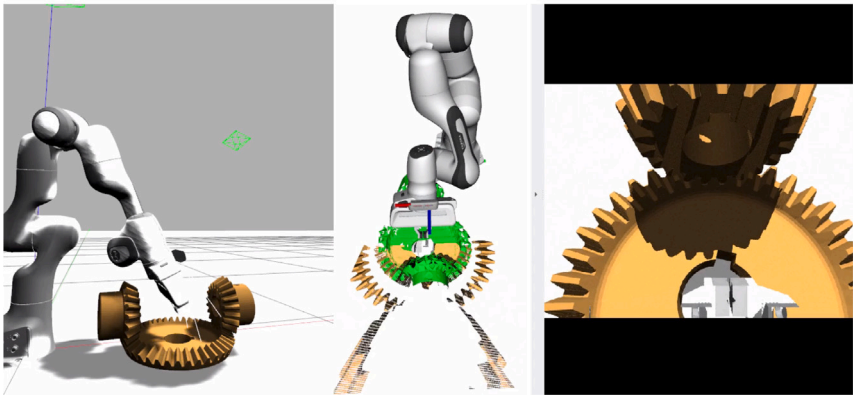
(a) Engine block under inspection



(b) Aircraft's right wing under inspection



(c) Gearbox under inspection



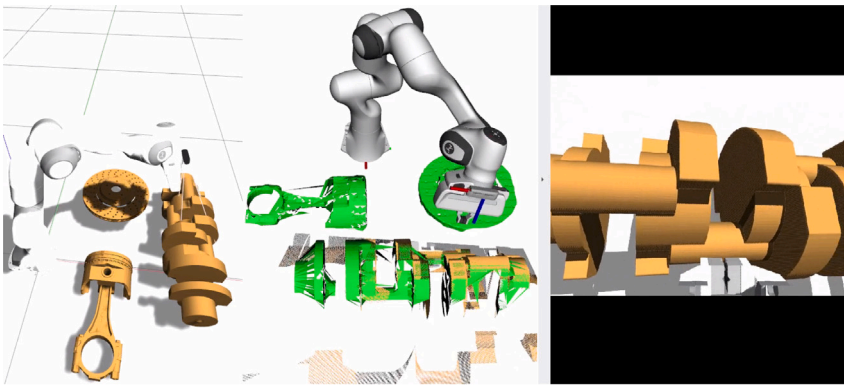
(d) A set of bevel gear under inspection

Fig. 4. Results of evaluation on single objects in simulation. The left image in each section is from Gazebo sim environment, the middle image depicts the overlayed collision avoidance mesh (green) along with generated inspection targets, and the right image show the real camera view.

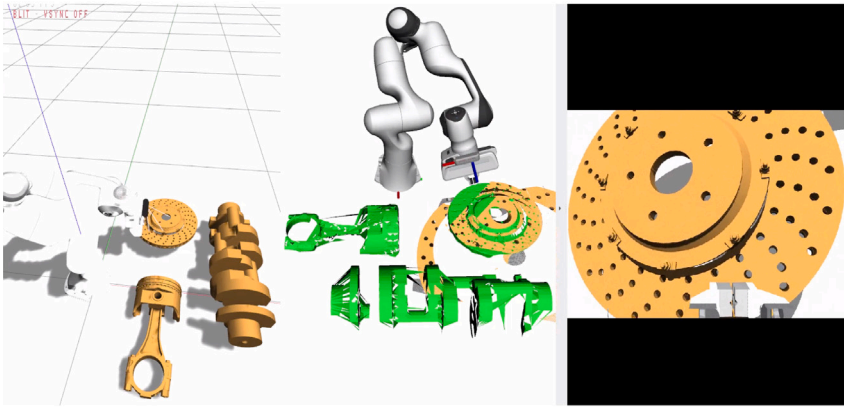
5.4. HRC based inspection in simulated environment

To perform a detailed evaluation of our natural language-based HRC system, we setup a simulated environment to perform inspection tasks on various industrial objects of varying complexities.

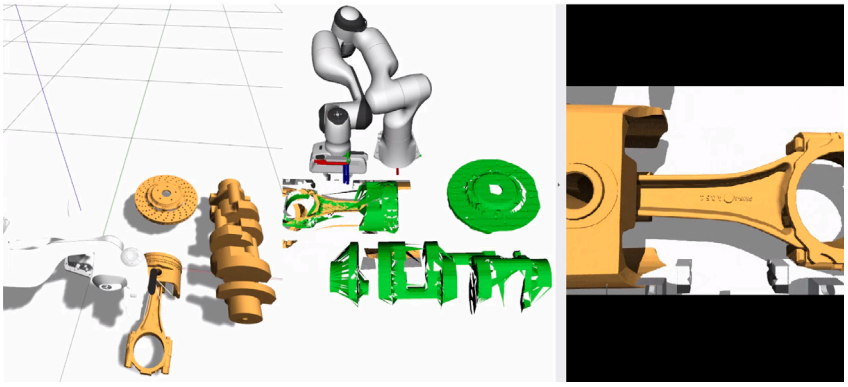
Single object inspection - The system was evaluated using a diverse set of realistic 3D models representing both small and large scale objects, including an engine block, an aircraft wing section, a transmission drive, and a set of bevel gears. The results of the simulation experiment are shown in Fig. 4. The robot assistant successfully interprets user instructions and performs inspections while effectively avoiding



(a) Crank shaft under inspection



(b) Disc brake under inspection



(c) Piston under inspection

Fig. 5. Results of evaluation on multiple objects in a cluttered simulation environment. The left image in each section is from Gazebo sim environment, the middle image depicts the overlayed collision avoidance mesh (green) along with generated inspection targets, and the right image show the real camera view.

collisions, as demonstrated in Figs. 4(a), 4(c), 4(d). The center image of Fig. 4(b) depicts the inspection targets generated based on the user's preferences.

Multi object inspection **-** To evaluate the performance of the system in cluttered environments, a simulated setup was constructed using multiple 3D object models, including a crankshaft (Fig. 5(a)), disc brake (Fig. 5(b)), and piston (Fig. 5(c)), positioned in close proximity to each other. The objects shown in Fig. 5 undergo visual inspection on individual objects while avoiding collisions.

5.5. HRC based inspection in real environment

To show the real-world capabilities of the system, we performed visual inspection on a dummy model of a suction roll as a proof of concept for the use case presented in Section 4. Fig. 6 shows the physical experimental setup. The left image shows the Franka robotic arm performing a visual inspection on a dummy suction roll. The center image displays the Rviz visualization, including the captured point cloud and the generated inspection targets. The right image presents the corresponding view from the end-effector-mounted camera.

5.6. Discussion

As a general result, the HRC system based on natural language can be used for visual inspection tasks and provides an much more natural and collaborative experience while providing better ergonomics for the user. The local deployment of the LLMs and the associated hardware preserves privacy and provides a functional system at a fraction of the cost.

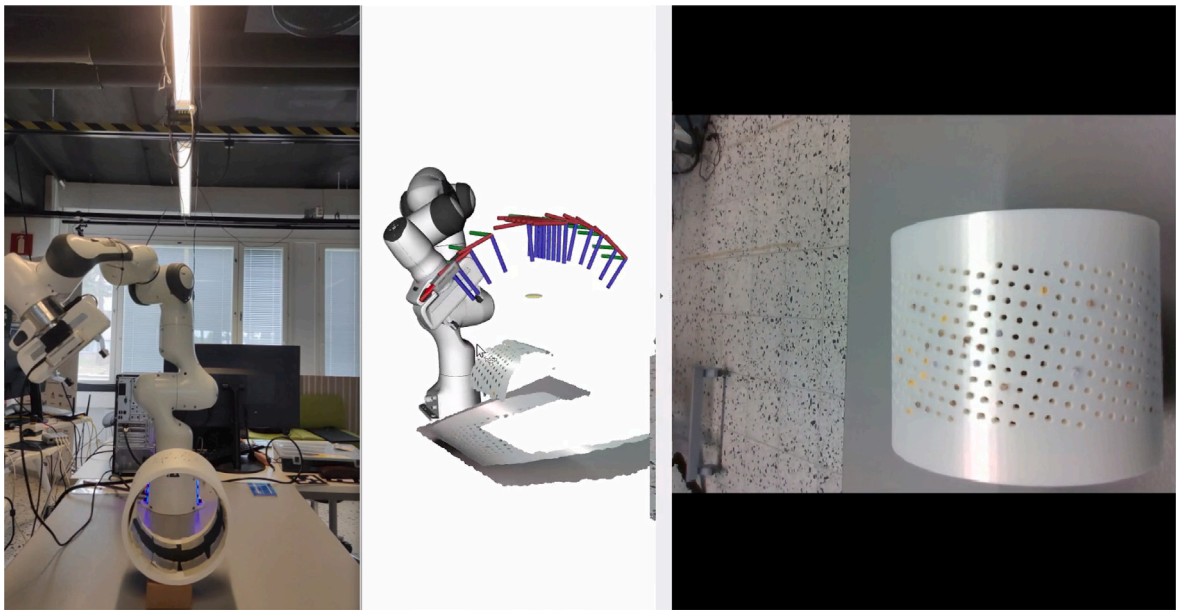


Fig. 6. Result of visual inspection performed on a dummy suction roll. The left image shows the physical setup with the robot and a dummy suction roll. The center image depicts the sim image from Rviz along with the generated targets. The image on the right shows the camera's perspective during the inspection process.

Model comparative - The results presented in Section 5.1 emphasize the critical role of selecting an appropriate LLM for effective deployment in resource-constrained environments. Models such as Deepseek-Coder-7B (Q8) [42] and Deepseek-Coder-1.3B-instruct [43] demonstrated greater difficulty in following prompts and generating accurate responses. While larger models tend to yield higher accuracy in inspection tasks, they incur significantly greater computational demands. Therefore, it is essential to select a model that achieves an optimal balance between response speed, accuracy, and resource efficiency. Overall, the use of Qwen2.5-Coder-1.5B-Instruct [35] provides a great balance between speed, accuracy, and resource usage for our use case. However, it should be noted that the Qwen2.5-Coder-7B-Instruct (Q8) [41] model provides the best value in terms of speed and accuracy with a reasonable compromise in resource usage.

Prompt adherence - The results in Section 5.2 highlight the importance of precise prompt formulation and the need for task-specific fine-tuning or prompt-engineering strategies to improve performance in low-scoring cases. Prompts such as *cluster point cloud* showed greater resistance to prompt variation than others. This may be due to suboptimal instruction formatting in the system prompt, where the code was presented as a single block. This issue can be mitigated by encapsulating the code within well-defined, descriptive functions in the system prompt, thereby enhancing interpretability and robustness. Furthermore, during experimentation, the prompt create robot targets consistently failed to generate a minor yet important code segment responsible for notifying the user of the current action being executed. While this exclusion does not affect the functionality of the system, it does diminish the quality of human–robot interaction for the given task. A likely explanation is the presence of biases in the training data of the LLM. This issue can be mitigated by adopting a few-shot prompting approach, where, some examples are included in the system prompt to guide the model toward producing more complete and context-aware responses.

Closed-loop execution efficiency - The results in Section 5.3 indicate that the total code inference time during closed-loop execution is significantly less than the overall task execution times for the inspected objects. This gap arises from multiple factors, including the processing power of the GPU hardware used for inference, default timeouts set in speech recognition and transcription, and user-configurable interaction settings with the robot assistant. These observations highlight that while model inference contributes only a small fraction of the total execution time, system-level and environmental variables play a major role in determining end-to-end task performance.

Visual inspection in sim and real environments - The results from Section 5.4 provides greater insights in the application of this system to perform visual inspection tasks on more complex objects and on objects placed in a cluttered environment. The robot assistant performed well in inspecting such objects due to the introduction of a new obstacle avoidance pipeline to the inspection workflow that allowed for collision-aware planning and robot motion. However, a few collisions were observed particularly with objects positioned in close proximity to the robot's base. This is an expected limitation, as such placement constrains the available workspace, restricting the robot's link movements and increasing the likelihood of collision.

The system was evaluated on several smaller objects, including a crankshaft, piston, and disc brakes, and demonstrated satisfactory performance even on items with complex geometries. The target generation algorithm relies on the object's geometry, that makes it generalizable and applicable to objects of varying sizes and shapes. However, because trajectory generation relies on the current viewpoint of the camera, partial occlusion can prevent accurate trajectory generation, as the depth camera depends on the visible geometry from its current perspective.

The Section 5.5 presented results for the industrial use case discussed in Section 4. Unlike simulated environments, the real environment presents various challenges, including variation in ambient conditions such as lighting and background noise that impacts the vision and speech systems. Though the neural speech processing models are quite robust to such variations, there were certain instances where the system misheard a command and pre-maturely triggers a response. Although this can be mitigated by fine-tuning the speech recognition model on specific user sound patterns on the fly, making it better immune to such anomalies.

Dynamic interaction with the system - The system takes commands from the user in a discontinuous manner, which gives it a high degree of flexibility and can be adapted to various situations. It also includes

several built-in safety features, including single-shot mesh-based obstacle avoidance, coordinate filters that prevent the robot's end-effector from entering hazardous regions by excluding certain targets based on user configurations, and an emergency stop function that allows the user to halt robot motion at any point during the execution by simply instructing the assistant to stop the robot. In addition, the user can command the robot to return to its initial position and restart the inspection process when necessary. With low-level control handled by ROS, the motion planner is able to generate feasible and deterministic trajectories. All commands are executed in a separate thread, allowing for parallel processing and significantly improve the response capability. These integrated safety features improves system robustness and allows for a more effective human–robot interaction. The overall response time is limited only by the underlying hardware used for LLM processing and the size of the deployed model, as discussed in Sections 5.1 and 5.3.

Limitations - The LLM-enabled human–robot collaboration system for visual inspection demonstrates robust performance across objects of varying shapes and dimensions, even in complex environments. However, its overall accuracy and responsiveness are constrained by the limited computational resources available at our disposal. Using more capable models could further enhance the quality of natural language interaction. Additionally, speech transcription accuracy is sometimes compromised, particularly in noisy environments or when multiple speakers are present. This limitation is partly due to the use of a compact speech recognition model. A potential solution may involve fine-tuning the model on domain-specific audio-text datasets containing technical instructions, along with feeding voice samples of the user on the fly. Such adaptations would improve the model's ability to capture nuanced speech patterns and enhance its robustness.

Integration effort - The resources and effort needed to develop, train and deploy perception models for industrial use, is considerable. Even when robust and reliable pre-trained models are to be integrated, still effort is needed to comply tools to existing software frameworks with its own datatypes and formatting. While ROS1 has taken first steps to enable this for robotics, LLM tools are typically disconnected from this.

5.7. Industrial deployment estimates

In general, the deployment costs scale proportionally to the size of the implementation. In this section, we provide realistic cost estimates for recommended configurations, along with reasonable projections of the system's speed and efficiency.

Computational Hardware - The system relies on GPU for running various LLM subsystems, including speech-to-text, code generation, and text-to-speech modules. Depending on the inspection scale and desired performance, different GPU configurations can be selected to balance cost and efficiency.

Baseline GPU (RTX 3060, 12 GB) - This GPU is no longer recommended, as it is two generations old. However, since all our experiments were conducted using this hardware, it serves as a baseline for comparison with other configurations. At an estimated cost of around 350 USD (2025), the RTX 3060 provided sufficient VRAM to run smaller LLM models, though its performance was relatively limited and slower compared to newer options. The costs and configurations listed are intended as reference points, with many other options available that may be better suited to specific industry requirements.

RTX 5090, 32 GB - This high-end consumer-grade GPU provides substantially higher inference speed and increased memory capacity, allowing for the deployment of larger and more optimized models while ensuring smoother closed-loop control. It is particularly well-suited for mid-sized industrial applications where both performance and reliability are critical. The RTX 5090 is estimated to deliver a 335% improvement in speed over the baseline RTX 3060 [47], making it more than three times faster. The estimated cost of this GPU is approximately 2000 USD.

PC and other components - A standard workstation with supporting components costs approximately 1,000–1,500 USD, regardless of GPU selection. Deploying a robotic work-cell for inspection tasks is estimated at around 100,000 USD, though actual costs vary depending on the robot model, payload capacity, and required end-effectors. All software, tools, and programs used with our robot assistant code are open source and do not require any additional licensing fees.

Operational Costs - These include both electricity and system maintenance. Annual electricity expenses are estimated at 500 USD per GPU. For maintenance, consumer-grade GPUs used in industrial workloads typically require replacement every 4 to 5 years, adding approximately 2,000 USD over a five year period.

Personnel Training - The proposed system is designed for intuitive interaction via natural language, reducing the need for extensive programming expertise. A training period of 1–2 weeks (orientation, demonstrations, safety training, and test runs) is considered sufficient for operators to effectively work with the robot assistant.

6. Conclusions

In this work, we presented Human–Robot Collaborative visual inspection with Large Language Models that offers an intuitive interface for communication through natural language. The system addressed some of the key challenges in modern industrial environments, including the need for intuitive interaction, data privacy, and resource efficiency. The hybrid architecture combines high-level instruction generation through LLMs with a robust perception pipeline and low-level motion control via ROS, ensuring both adaptability and safety. The system proved effective in both simulated and real-world environments based on industrial use cases, demonstrating its ability to manage complex inspection tasks involving objects of varying sizes and structural complexities. However, through extensive experiments, we also noted the shortcomings in speech transcription accuracy that is sometimes compromised in noisy environments or in the presence of multiple speakers due to the use of a compact speech recognition model and provided multiple potential solutions to overcome such situations. Furthermore, we observed that the accuracy of the system depended on the size of the model used. Larger models often produce more accurate results, but at the cost of more resources. Hence, a trade-off was established between speed, accuracy, and resource consumption. Overall, this approach highlights the potential of combining natural language, generative AI, and robotics to advance collaborative, human-centered automation in manufacturing.

Future work - Future work will involve the implementation of vision-based LLMs with hybrid control for industrial tasks involving inspection, picking and placing, sorting, and task planning.

CRediT authorship contribution statement

Osama Tasneem: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Roel Pieters:** Writing – review & editing, Supervision, Project administration, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This project has received funding from the European Union's Horizon Europe research and innovation programme under Grant Agreement no. 101059903 and 101135708. In addition, we acknowledge financial support of the Finnish Ministry of Education and Culture through the Intelligent Work Machines Doctoral Education Pilot Program (IWM VN/3137/2024-OKM-4).

Data availability

Data will be made available on request.

References

[1] International Federation of Robotics, World robotics 2023: Industrial robots, 2023, statistical Department Report. https://ifr.org.

[2] M. Dhanda, B.A. Rogers, S. Hall, E. Dekoninck, V. Dhokia, Reviewing human–robot collaboration in manufacturing: Opportunities and challenges in the context of industry 5.0, Robot. Comput. Integr. Manuf. 93 (2025) 102937, http://dx.doi.org/10.1016/j.rcim.2024.102937.

[3] T. Lohi, S. Soutukorva, T. Heikkilä, Programming of skill-based robots, in: IEEE 19th Conference on Industrial Electronics and Applications, ICIEA, 2024, pp. 1–7, http://dx.doi.org/10.1109/ICIEA61579.2024.10664981.

[4] C. Schou, R.S. Andersen, D. Chrysostomou, S. Bøgh, O. Madsen, Skill-based instruction of collaborative robots in industrial settings, Robot. Comput.-Integr. Manuf. (2018) http://dx.doi.org/10.1016/J.RCIM.2018.03.008.

[5] T.B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, et al., Language models are few-shot learners, In: Int. Conf. Neural Inf. Process. Syst. 187 (2020) 7–1901.

[6] O. Tasneem, R. Pieters, Automatic robot path planning for active visual inspection on free-form surfaces, in: IEEE 20th International Conference on Automation Science and Engineering, CASE, 2024, pp. 173–180, http://dx.doi.org/10.1109/CASE59546.2024.10711320.

[7] International Federation of Robotics, LBR iisy cobot in quality assurance: Grinding coffee to the exact gram, 2024b, https://ifr.org/case-studies/lbr-iisy-cobot-in-quality-assurance.

[8] International Federation of Robotics, Collaborative assembly application with HC10, 2024a, https://ifr.org/case-studies/collaborative-assembly-application-with-hc10.

[9] L. Gualtieri, E. Rauch, R. Vidoni, Emerging research fields in safety and ergonomics in industrial collaborative robotics: A systematic literature review, Robot. Comput. Integr. Manuf. 67 (2021) 101998, http://dx.doi.org/10.1016/j.rcim.2020.101998.

[10] M. Lorenzini, M. Lagomarsino, L. Fortini, S. Gholami, A. Ajoudani, Ergonomic human–robot collaboration in industry: A review, Front. Robot. AI 9 (2023) http://dx.doi.org/10.3389/frobt.2022.813907.

[11] V. Villani, F. Pini, F. Leali, C. Secchi, Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications, Mechatronics (2018) http://dx.doi.org/10.1016/J.MECHATRONICS.2018.02.009.

[12] IKEI EU-OSHA, J.d Panteia Kok, P. Vroonhof, J. Snijders, G. Roullis, M. Clarke, K. Peereboom, P.v. Dorst, I. Isusi, Work-Related Musculoskeletal Disorders – Prevalence, Costs and Demographics in the EU, Publications Office, 2019, http://dx.doi.org/10.2802/66947.

[13] Z. Lončarević, A. Gams, S. Reberšek, B. Nemec, J. Škrabar, A. Ude, Specifying and optimizing robotic motion for visual quality inspection, Robot. Comput.-Integr. Manuf. 72 (2021) 102200, http://dx.doi.org/10.1016/j.rcim.2021.102200.

[14] D. Mukherjee, K. Gupta, L.H. Chang, H. Najjaran, A survey of robot learning strategies for human–robot collaboration in industrial settings, Robot. Comput.-Integr. Manuf. 73 (2022) 102231, http://dx.doi.org/10.1016/j.rcim.2021.102231.

[15] M. Cakmak, A.L. Thomaz, Designing robot learners that ask good questions, in: 2012 7th ACM/IEEE International Conference on Human-Robot Interaction, HRI, 2012, pp. 17–24, http://dx.doi.org/10.1145/2157689.2157693.

[16] D. Koch, J.P. Kaiser, F. Stamer, R. Stark, G. Lanza, Enhancing visual inspection in remanufacturing: A reinforcement learning approach with integrated robot simulation, Procedia CIRP 134 (2025) 939–944, http://dx.doi.org/10.1016/j.procir.2025.02.228.

[17] H. Zhang, S.D. Semujju, Z. Wang, X. Lv, K. Xu, L. Wu, et al., Large scale foundation models for intelligent manufacturing applications: a survey, J. Intell. Manuf. (2025) http://dx.doi.org/10.1007/s10845-024-02536-7.

[18] S. Zhao, S. Liu, Y. Jiang, B. Zhao, Y. Lv, J. Zhang, et al., Industrial foundation models (ifms) for intelligent manufacturing: A systematic review, J. Manuf. Syst. 82 (2025) 420–448, http://dx.doi.org/10.1016/j.jmsy.2025.06.011.

[19] C. Matuszek, E.V. Herbst, L. Zettlemoyer, D. Fox, Learning to parse natural language commands to a robot control system, in: International Symposium on Experimental Robotics, 2013, pp. 403–415, http://dx.doi.org/10.1007/978-3-319-00065-7_28.

[20] D. Misra, J. Sung, K. Lee, A. Saxena, Tell me dave: Context-sensitive grounding of natural language to manipulation instructions, Int. J. Robot. Res. 35 (2014) 281–300, http://dx.doi.org/10.1177/0278364915602060.

[21] T. Kollar, S. Tellex, D.K. Roy, N. Roy, Toward understanding natural language directions, ACM/IEEE Int. Conf. Human Robot Interact. (HRI) 25 (2010) 9–266, http://dx.doi.org/10.1145/1734454.1734553.

[22] J. Thomason, S. Zhang, R.J. Mooney, P. Stone, Learning to interpret natural language commands through human–robot dialog, in: 24th International Conference on Artificial Intelligence, 2015, pp. 1923–1929.

[23] A. Vaswani, N.M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, Neural Inf. Process. Syst. (2017) 6000–6010, https://dl.acm.org/doi/10.5555/3295222.3295349.

[24] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. Pondé, J. Kaplan, et al., Evaluating large language models trained on code, 2021, ArXiv:abs/2107.03374.

[25] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, et al., . Qwen2.5-coder technical report., 2024e, arXiv preprint arXiv:2409.12186.

[26] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. Tan, et al., Code Llama: Open foundation models for code, 2023, ArXiv:abs/2308.12950.

[27] S.H. Vemprala, R. Bonatti, A.F.C. Bucker, A. Kapoor, ChatGPT for robotics: Design principles and model abilities, IEEE Access 12 (2023) 55682–55696, http://dx.doi.org/10.1109/ACCESS.2024.3387941.

[28] M.G. Arenas, T. Xiao, S. Singh, V. Jain, A. Ren, Q. Vuong, et al., How to prompt your robot: A promptbook for manipulation skills with code as policies, IEEE Int. Conf. Robot. Autom. (ICRA) 434 (2024) 0–4348, http://dx.doi.org/10.1109/ICRA57147.2024.10610784.

[29] A. Radford, J.W. Kim, T. Xu, G. Brockman, C. McLeavey, I. Sutskever, Robust speech recognition via large-scale weak supervision, in: 40th International Conference on Machine Learning, 2023, pp. 28492–28518.

[30] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, K. Choromanski, T. Ding, et al., RT-2: vision-language-action models transfer web knowledge to robotic control, 2023, ArXiv:abs/2307.15818.

[31] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, et al., RT-1: Robotics transformer for real-world control at scale, 2022, ArXiv:abs/2212.06817.

[32] M. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, et al., OpenVLA: An open-source vision-language-action model, 2024, arXiv preprint arXiv:2406.09246.

[33] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, et al., Code as policies: Language model programs for embodied control, IEEE Int. Conf. Robot. Autom. (ICRA) 949 (2022) 3–9500, http://dx.doi.org/10.1109/ICRA48891.2023.10160591.

[34] H. Liu, Y. Zhu, K. Kato, A. Tsukahara, I. Kondo, T. Aoyama, Y. Hasegawa, Enhancing the LLM-based robot manipulation through human–robot collaboration, IEEE Robot. Autom. Lett. 9 (2024) 6904–6911, http://dx.doi.org/10.1109/LRA.2024.3415931.

[35] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Dang, et al., Qwen/qwen2.5-coder-1.5b-instruct, 2024b, https://huggingface.co/Qwen/Qwen2.5-Coder-1.5B-Instruct.

[36] Hexgrad, Hexgrad/kokoro-82m, 2025, https://huggingface.co/hexgrad/Kokoro-82M.

[37] Y.A. Li, C. Han, V.S. Raghavan, G. Mischler, N. Mesgarani, StyleTTS 2: Towards human-level text-to-speech through style diffusion and adversarial training with large speech language models, Adv. Neural Inf. Process. Syst. 36 (2023) 19594–19621.

[38] I.A. Sucan, M. Moll, L.E. Kavraki, The open motion planning library, IEEE Robot. Autom. Mag. 19 (2012) 72–82, http://dx.doi.org/10.1109/MRA.2012.2205651.

[39] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Dang, et al., Qwen/qwen2.5-1.5b-instruct, 2024a, https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct.

[40] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Dang, et al., Qwen/qwen2.5-coder-3b-instruct, 2024c, https://huggingface.co/Qwen/Qwen2.5-Coder-3B-Instruct.

[41] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Dang, et al., Qwen/qwen2.5-coder-7b-instruct-gguf, 2024d, https://huggingface.co/Qwen/Qwen2.5-Coder-7B-Instruct-GGUF.

[42] mradermacher, Mradermacher/tablellm-deepseekcoder-7b-gguf, 2024b, https://huggingface.co/mradermacher/TableLLM-DeepseekCoder-7B-GGUF.

[43] DeepSeek, Deepseek-ai/deepseek-coder-1.3b-instruct, 2024, https://huggingface.co/deepseek-ai/deepseek-coder-1.3b-instruct.

[44] second state, Second-state/deepseek-coder-v2-lite-instruct-gguf, 2024, https://huggingface.co/second-state/DeepSeek-Coder-V2-Lite-Instruct-GGUF.

[45] mradermacher, Mradermacher/dolphincoder-starcoder2-15b-i1-gguf, 2024a, https://huggingface.co/mradermacher/dolphincoder-starcoder2-15b-i1-GGUF.

[46] Q.Y. Zhou, J. Park, V. Koltun, Open3D: A modern library for 3d data processing, 2018, http://arxiv.org/abs/1801.09847.[arXiv:1801.09847].

[47] UserBenchmark, Benchmark comparative results between rtx 3060 and rtx 5090, 2025, https://gpu.userbenchmark.com/Compare/Nvidia-RTX-3060-vs-Nvidia-RTX-5090/4105vs4180https://gpu.userbenchmark.com/Compare/Nvidia-RTX-3060-vs-Nvidia-RTX-5090/4105vs4180.