

Trustworthy Neuro-Symbolic Agents in Manufacturing: Bridging CAD and RAPID through Local LLMs and Multi-Agent Orchestration

Alessandro Venaruzzo^{1,*}, Matteo Cristani¹

¹Department of Computer Science, University of Verona, Strada Le Grazie 15, 37134 Verona, Italy

Abstract

The transition from engineering drawings to precise manufacturing recipes traditionally demands time-consuming manual intervention and specialized domain expertise. While Large Language Models (LLMs) offer compelling generative capabilities, ensuring accuracy, traceability, and determinism remains a paramount challenge in intelligent manufacturing. In this paper, we propose a framework for autonomous recipe synthesis based on an architecture governed by three core layers: a local LLM, Retrieval-Augmented Generation (RAG), and a Knowledge Graph (KGraph). The KGraph provides scaffolding for classification and decision-making, grounding the generative process in structural industrial realities. We present a conceptual architecture and validate it through a limited pilot implementation of a modular multi-agent orchestration encompassing metadata identification, batch comparison, historical retrieval, validation, and scratch generation for instances lacking prior transformation examples.

We use the term multi-agent in the pragmatic sense of specialized agentic modules orchestrated sequentially, rather than in the classical sense of decentralized autonomous agents: each agent denotes a specialized functional module performing a well-defined transformation within the pipeline. Ultimately, we argue that orchestrating LLM flexibility, RAG contextual memory, and KGraph structural rigor provides a reliable paradigm for automated recipe synthesis in modern production scenarios, aligned with cyber-physical and digital twin paradigms.

Keywords

Neuro-Symbolic AI, Trustworthy AI, Industrial Agents, Large Language Models, Industrial Robotics, Safety-Critical Systems

1. Introduction

This work investigates the following research question: *Can a neuro-symbolic architecture combining LLMs, retrieval mechanisms, and structured knowledge enforce deterministic safety constraints in industrial recipe synthesis?*

The modern manufacturing paradigm, often framed within the context of Industry 5.0 [1], increasingly prioritizes the integration of resilient, intelligent, and knowledge-driven technologies. A critical bottleneck in this domain remains the extraction of implicit knowledge from manufacturing drawings and its subsequent transformation into actionable production recipes, such as those written in RAPID, the proprietary high-level programming language used for ABB industrial robots. This translation process traditionally relies heavily on domain experts, making it time-intensive and highly susceptible to variations in expert judgment. In this context, a “recipe” denotes a structured and executable set of parameters and instructions governing an industrial production process.

This research is conducted in collaboration with EssilorLuxottica, a global leader in eyewear manufacturing. The project is motivated by industrial needs in the automation domain, particularly the reduction of the manual, repetitive effort required to reconfigure production systems across varying batches. Such challenges arise in complex environments involving robotic systems, PLCs, and CNC machines developed in-house. Within this context, the project focuses on leveraging data-driven methods to improve efficiency and adaptability. The broader research project also addresses the scarcity of

WOA 2026: 27th Workshop From Objects to Agents, June 15–17, 2026, Salerno, Italy

*Corresponding author.

✉ alessandro.venaruzzo@univr.it (A. Venaruzzo); matteo.cristani@univr.it (M. Cristani)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

annotated data typical of rapidly changing industrial scenarios (e.g., through synthetic data generation for industrial object detection); the present paper focuses on the recipe synthesis component.

While Large Language Models (LLMs) have demonstrated remarkable capabilities in generalized zero-shot generation and text-to-action behaviors [2, 3], their standalone deployment in industrial engineering introduces significant vulnerabilities. The generative nature of these models often results in plausible yet factually incorrect outputs, a phenomenon incompatible with the rigorous deterministic requirements of factory environments. Therefore, there is a fundamental need for hybrid strategies that merge neural semantic flexibility with robust symbolic and contextual anchoring mechanisms.

Focusing on scalable and trustworthy recipe synthesis, this paper proposes an architecture based on three interconnected layers: (1) a foundational LLM providing the core semantic processing capacity; (2) a Retrieval-Augmented Generation (RAG) module built on pairs of historical drawings and corresponding recipes, where RAG denotes a paradigm in which external knowledge is retrieved and injected into the generative process to improve factual grounding [4]; and (3) a lightweight property-based Knowledge Graph (KGraph) [5] dedicated to structured classification, identification, and deterministic decision-making support, in line with recent work on industrial knowledge graphs for manufacturing reasoning [6, 7].

The core objective of the proposed system is the automatic generation of industrial recipes (for instance, structured parameter sets for PLC/CNC-controlled processes) starting from engineering design data. In particular, the system takes as input a DWG (Drawing) file, a standard CAD format developed by Autodesk, together with associated metadata, and produces a validated set of process parameters capable of manufacturing components consistent with the input design.

In this paper, RAPID code generation is treated as a representative case of the broader recipe synthesis problem, where recipes denote structured executable instructions or parameter configurations for industrial automation systems.

The task can be understood as a multi-stage transformation pipeline in which raw geometric and contextual information is progressively enriched and translated into executable production knowledge. More specifically, the system addresses the following sub-problems:

- **Geometric Interpretation:** Extraction and analysis of relevant features from the DWG file, including shapes, dimensions, tolerances, and structural relationships between components.
- **Context Construction:** Integration of metadata and inferred information to define the production context. This includes:
 - identification of the *type of production* (e.g., machining, assembly, additive manufacturing),
 - characterization of the *target device or machine class* (e.g., CNC milling machine, robotic cell, PLC-controlled line),
 - recognition of constraints related to materials, precision, and operational requirements.
- **Knowledge Grounding:** Alignment of the interpreted design and contextual information with domain knowledge sources (e.g., historical recipes, rules, and ontologies), enabling the system to map abstract design intent to feasible production strategies.
- **Recipe Synthesis:** Generation of a candidate recipe, defined as a structured set of parameters and instructions governing the production process. These may include:
 - machine configurations,
 - process parameters (e.g., speeds, feeds, temperatures),
 - sequencing of operations,
 - control logic for PLC/CNC execution.
- **Validation and Consistency Checking:** Verification that the generated recipe is coherent with domain constraints, machine capabilities, and the intended production outcome, ensuring both feasibility and correctness.
- **Output Structuring:** Formatting of the final recipe into a representation directly usable by downstream systems, such as industrial controllers or simulation environments.

Figure 1 provides an illustrative overview of the target problem addressed in this work, namely the automatic generation of industrial recipes from engineering design data. The figure is structured into multiple panels, each highlighting a key stage of the transformation pipeline, from geometric input to executable production parameters.

On the left-hand side (Panel 1), a representative example of a DWG file is shown in its PNG-rendered form. The drawing corresponds to a mechanical component, specifically a flanged bushing, characterized by a combination of external diameters, internal precision bores, and a flange with equally spaced through-holes. The drawing includes dimensional constraints, tolerances (e.g., H7 fits), surface finish specifications, and material information. This type of component is typical of parts manufactured using CNC turning centers equipped with automated reaming operations, and therefore constitutes a representative example for illustrating the proposed approach.

Panel 2 provides a structured interpretation of the drawing. From the raw geometric representation, a set of semantic features is identified, including the overall geometry of the part, the presence of precision bores requiring reaming, and the flange geometry with its bolt-circle pattern. In addition, contextual information is inferred, such as the likely production process (turning with reaming), the class of machine tool required, and the associated manufacturing constraints. This step exemplifies the transition from purely geometric data to enriched contextual knowledge.

Panel 3 presents an example of a corresponding industrial recipe. The recipe is expressed as a structured set of parameters, including machine type, tooling configuration (e.g., reamers for specific tolerances), spindle speeds, feed rates, coolant specifications, and inspection requirements. These parameters collectively define the operational conditions under which the part can be manufactured in compliance with the specifications extracted from the drawing.

Panel 4 illustrates the execution of the recipe on a representative machine setup. The sequence of images depicts the main stages of the machining process, including part clamping, drilling of a pilot hole, precision reaming of the bore, and finishing operations. This panel emphasizes the direct correspondence between abstract recipe parameters and their physical realization in an automated production environment.

Finally, Panel 5 compares the traditional manual workflow with the proposed automated approach. In the manual setting, an expert operator must interpret the drawing, select tools, compute process parameters, and iteratively validate the resulting program, typically requiring several hours of effort. In contrast, the proposed system aims to generate a validated recipe automatically, reducing this effort to minutes by leveraging integrated reasoning, retrieval, and knowledge-based validation mechanisms.

Overall, Figure 1 highlights the central challenge addressed in this work: bridging the gap between engineering design and executable manufacturing knowledge. The transformation from DWG input to validated recipe involves not only geometric interpretation, but also contextual understanding, domain-specific reasoning, and constraint enforcement. This motivates the need for a multi-layer architecture capable of combining generative intelligence with structured knowledge representations, as discussed in the following sections.

To address these challenges, we propose a trustworthy, hybrid neuro-symbolic agent architecture for automated CAD-to-RAPID programming. Neuro-symbolic artificial intelligence [8] advocates the synthesis of neural learning capabilities (for pattern recognition and semantics) with symbolic logic (for deterministic rule execution and safety constraints). This combination is especially relevant in safety-critical environments, where errors translate into physical risk [9, 10].

To operationalize these cognitive layers, we introduce a modular multi-agent pipeline. This framework breaks down the recipe generation pipeline into five cooperating modules: Metadata Identifier, Batch Comparator, RAG-Retrieval Module, Validation/Adequacy Module, and Scratch Reasoning & Generator Module. This pipelined approach ensures that new recipes are computationally synthesized through a series of evaluated, verifiable steps.

Through this architectural blueprint, illustrated by three diagrams (the general architecture in Figure 2, the multi-agent interaction in Figure 3, and the decision flow in Figure 4), we present a viable pathway toward reliable and explainable generative engineering in automated manufacturing.

Specifically, the proposed agent combines two complementary components:

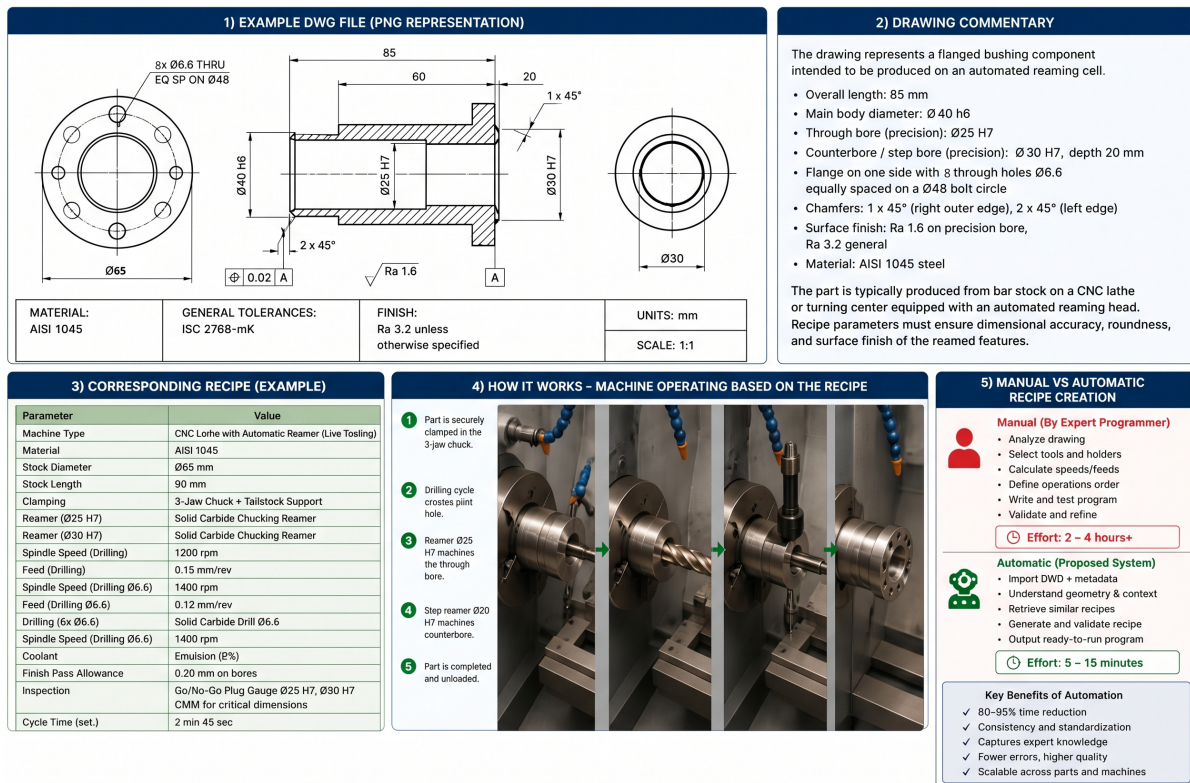


Figure 1: Illustrative example of the automatic generation of an industrial recipe from a DWG drawing, including geometric interpretation, contextualization, recipe synthesis, and execution.

- 1. Symbolic/Deterministic Extraction:** A geometric calculation pipeline that parses CAD files into a canonical payload, guaranteeing that the mathematical foundation of the robotic paths remains strictly exact and safe.
- 2. Neural/Semantic Reasoning:** A locally hosted LLM (e.g., Qwen2.5 served through Ollama) acting strictly at the “edges” of the process to resolve nomenclature ambiguities and map complex features (e.g., using a `machine_map.yaml` configuration) without exposing data off-site, consistently with emerging Edge AI paradigms for privacy-preserving inference [11, 12].

The proposed system is organized as a three-layer architecture composed of (i) a Large Language Model (LLM) layer, (ii) a Retrieval-Augmented Generation (RAG) layer, and (iii) a Knowledge Graph (KGraph) layer. The overall objective of this architecture is to support the generation of industrial recipes, intended as structured parameter sets for automation systems (e.g., PLC and CNC configurations), by combining data-driven reasoning with structured domain knowledge and contextual awareness.

At the foundational level, the LLM constitutes the core reasoning engine of the system. It provides natural language understanding, abstraction, and generative capabilities, enabling the interpretation of user queries and the formulation of candidate solutions. All agentic reasoning activities are grounded in this component, which is responsible for synthesizing information and producing intermediate and final outputs. However, while the LLM offers strong generalization and inference capabilities, it operates without intrinsic domain constraints, thus requiring complementary mechanisms to ensure contextual grounding and correctness.

The second layer implements a Retrieval-Augmented Generation (RAG) framework, whose primary role is to enhance the LLM with domain-specific knowledge. This layer is built on a curated dataset consisting of paired representations of engineering drawings and corresponding recipes, thereby establishing a mapping between geometric or structural information and parameter configurations. Due to industrial confidentiality constraints, the dataset cannot be publicly released. However, the pipeline structure, module interactions, and evaluation criteria are fully described to ensure conceptual

reproducibility.

During inference, the RAG module retrieves relevant instances and contextual information, which are injected into the generative process to guide and constrain the LLM outputs. In addition, the RAG layer integrates a reasoning-chain component based on generative code technologies (e.g., program-aided reasoning and code-generation tools). This mechanism enables the decomposition of complex tasks into structured intermediate steps, improving both the interpretability and reliability of the reasoning process.

The third layer, the Knowledge Graph (KGraph), provides a formal and structured representation of domain knowledge. It encodes entities, relationships, constraints, and semantic roles relevant to the application domain, thereby supporting higher-level reasoning tasks such as classification, entity linking, ambiguity resolution, and consistency checking. Beyond serving as a passive knowledge repository, the KGraph plays an active role in decision-making by enforcing domain-specific rules and validating the coherence of generated outputs. Furthermore, it interacts bidirectionally with both the RAG and LLM layers, enriching the reasoning process with structured semantic information and enabling the alignment of generated content with domain ontologies.

From an operational perspective, when an agent is tasked with generating a recipe, the system performs a multi-stage synthesis process that leverages all three layers in a coordinated manner. The LLM generates candidate parameter configurations based on the input query and its internal reasoning capabilities. Simultaneously, the RAG layer retrieves relevant contextual knowledge and injects it into the generation pipeline, while also orchestrating reasoning chains to structure the inference process. The KGraph layer subsequently contributes to classification, identification, and validation tasks, ensuring that the generated outputs satisfy domain constraints and exhibit semantic consistency. This integrated, agentic workflow results in context-aware, validated, and interpretable recipes, combining the flexibility of generative models with the rigor of structured knowledge representations [13].

The cornerstone of this architecture is the **Trustworthy Quality Gate**, a deterministic validation module, hereafter referred to as the “Quality Gate”. Rather than allowing the LLM to directly emit control actions [14], the Quality Gate acts as an algorithmic decision boundary. It evaluates the semantic proposals against verifiable spatial data: if all constraints are satisfied and the confidence threshold is met, the system proceeds autonomously (OK). If a constraint violation or insufficient confidence is detected, the pipeline first attempts a bounded retry; if the issue persists, the agent assumes a *HOLD* state and generates an explainable diagnostic report for operator intervention (Human-in-the-Loop).

This paper outlines the conceptual design of this agent and demonstrates how separating geometric rigor from semantic flexibility offers a reproducible blueprint for reliable AI in Industry 5.0. While the introductory example uses DWG as a common CAD representation, the pilot implementation operates on RSPAG archives (RobotStudio Pack&Go, a proprietary industrial archive format containing structured geometric and metadata information), here considered as equivalent structured engineering inputs.

The remainder of this paper is structured as follows. Section 2 reviews the relevant background on LLM-based robotic programming, trustworthy AI in Industry 5.0, and neuro-symbolic approaches. Section 3 introduces the proposed layered cognitive architecture, detailing the interaction between the LLM, RAG, and Knowledge Graph components. Section 4 presents the modular multi-agent workflow, describing the sequential orchestration of the system modules and the role of the Quality Gate. Section 5 reports the experimental case study and the baseline comparison with a pure LLM approach. Section 6 discusses the main results, including the trade-off between latency, syntactic validity, and geometric safety. Finally, Section 7 concludes the paper and outlines directions for future work.

2. Background and Related Work

The proposed hybrid architecture lies at the intersection of Large Language Models (LLMs) for embodied control, Trustworthy AI in Industry 5.0, and neuro-symbolic reasoning frameworks. We first review LLM-based code generation; we then discuss LLMs in robotic programming and the related privacy

constraints, Trustworthy AI in Industry 5.0, and finally neuro-symbolic agent architectures, in light of recent developments in LLM-based agentic systems [3, 15].

2.1. LLMs for Code Synthesis and Domain-Specific Generation

Large Language Models have recently reshaped the field of program synthesis by enabling the generation of executable code from natural-language specifications. Early systems such as Codex demonstrated that transformer models fine-tuned on source code can synthesize functionally correct programs from docstrings, introducing benchmarks such as HumanEval to evaluate code generation through execution-based correctness rather than surface-level similarity [16]. Subsequent work extended this paradigm toward more complex programming tasks. AlphaCode showed that large-scale sampling, filtering, and ranking can achieve competitive performance on programming-contest problems [17], while CodeGen introduced open large language models for multi-turn program synthesis, emphasizing the importance of decomposing programming tasks into sequential conversational steps [18]. More recent code-oriented models, such as CodeT5+, further highlight the relevance of specialized pretraining objectives and architectures for code understanding and generation across multiple downstream tasks [19]. Comprehensive surveys confirm that LLM-based code generation has become a mature research area, covering model architectures, training corpora, evaluation protocols, and practical limitations [20].

The problem addressed in this paper is related to this literature because manufacturing recipe generation can be interpreted as a form of domain-specific code synthesis: the system must transform an engineering specification into a structured, executable artifact. In this sense, the task is simpler than general-purpose software synthesis, since the output space is constrained by industrial schemas, machine parameters, and admissible transformation rules. Unlike open-ended programming, the generated recipe is not expected to implement arbitrary algorithms, but to instantiate a bounded family of manufacturing procedures.

At the same time, the task is more demanding than conventional code synthesis in several respects. First, correctness cannot be assessed only through syntactic validity or unit tests; it must also satisfy geometric, physical, and operational constraints imposed by the manufacturing environment. Second, the input is not merely a textual requirement, but a heterogeneous engineering artifact composed of CAD geometry, metadata, tolerances, historical recipes, and machine-specific constraints. Third, hallucinated outputs are not simply software bugs, but may correspond to unsafe or infeasible production actions. For this reason, our work departs from purely neural code generation and adopts a neuro-symbolic architecture in which LLM-based synthesis is constrained by retrieval, knowledge-graph grounding, and deterministic validation.

2.2. LLMs in Robotic Programming and Privacy Constraints

The advent of LLMs has significantly impacted automated code generation. Recent literature explores the application of these models to robotics, demonstrating their ability to translate high-level semantics into executable control policies [21]. Microsoft’s research on applying foundation models to robotics [2] highlights the potential of LLMs to generate complex routines when appropriately constrained by specialized APIs.

However, the overwhelming majority of these approaches rely on cloud-hosted foundation models accessed via third-party APIs. In the context of industrial automation, this represents a critical vulnerability. Exposing proprietary Computer-Aided Design (CAD) metadata, nomenclature, and plant-specific mapping rules (e.g., `machine_map.yaml` configurations) to external servers violates stringent corporate intellectual property (IP) policies. Consequently, there is an urgent need for *Edge AI* architectures, such as locally hosted models like Qwen2.5 [22] served via the Ollama framework [23], that can perform semantic reasoning without data exfiltration.

2.3. Trustworthy AI in Industry 5.0

Industry 5.0 shifts the focus from pure automation to human-centric, resilient, and sustainable manufacturing, heavily emphasizing the concept of *Trustworthy AI*. When generative models are applied to physical systems, the tolerance for “hallucinations” (plausible but factually incorrect outputs, a well-documented limitation of large language models [24]) drops to zero. A hallucinated coordinate or misclassified part feature can result in catastrophic hardware collisions.

To mitigate this, approaches like SayCan [14] have proposed “grounding” language model outputs in robotic affordances, effectively filtering what the AI *wants* to do through the lens of what the robot *can safely* do. While robust, these methods still largely treat the control loop as a unified neural or probabilistic problem. Our work extends this philosophy to the industrial CAD-to-RAPID pipeline by introducing a deterministic Quality Gate that acts as an absolute safeguard, preventing any probabilistically generated output from reaching the robotic controller without mathematical verification.

2.4. Neuro-Symbolic Agent Architectures

The necessity of combining statistically driven neural reasoning with rigid logic has led to the resurgence of Neuro-Symbolic AI, often referred to as the “Third Wave” of Artificial Intelligence [8]. Neuro-symbolic systems aim to retain the semantic flexibility of neural networks while inheriting the explainability and verifiability of symbolic logic.

In the domain of robotic path generation, extracting geometries from formats like RSPAG requires absolute deterministic precision (the symbolic component). Conversely, resolving the inherent ambiguities in human-defined CAD naming conventions requires cognitive flexibility (the neural component). Existing literature acknowledges the theoretical power of hybrid architectures, but practical, production-ready blueprints for industrial manufacturing remain scarce. By explicitly isolating the .NET-based geometric extraction pipeline from the LLM-based semantic mapping module, our proposed agent provides a measurable, explainable, and inherently safe implementation of a neuro-symbolic system tailored for reliable industrial operations.

3. Proposed Architecture: A Layered Cognitive Framework

The proposed system adopts a layered cognitive architecture designed to balance semantic flexibility with deterministic industrial reasoning. A core innovation of this paradigm is complementing deterministic geometric extraction with a hybrid AI integration for the synthesis of actionable recipes. To ensure awareness in decision-making and robustness in reasoning, the architecture is structured into three distinct layers, as illustrated in the general architecture diagram (Figure 2): (1) Large Language Model (LLM), (2) Retrieval-Augmented Generation (RAG), and (3) Knowledge Graph (KGraph).

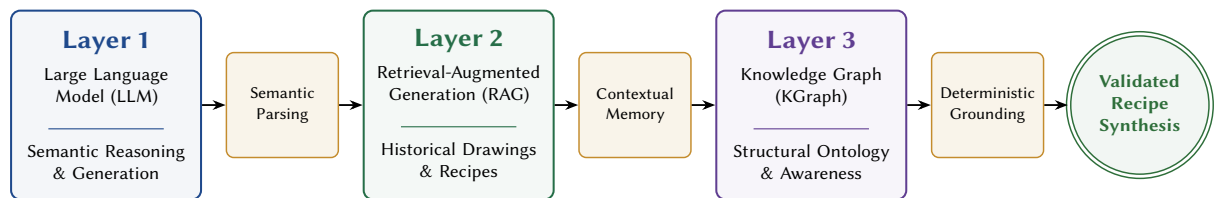


Figure 2: High-level architectural flow depicting the interplay of the LLM, RAG, and KGraph layers to synthesize recipes from input drawings.

3.1. Layer 1: Large Language Model (LLM)

The foundational computational layer relies on an edge-deployed LLM (e.g., Qwen2.5 [22] via the Ollama framework [23]). By operating locally, this layer prevents sensitive proprietary layouts from

being externalized, meeting the strict confidentiality requirements of industrial settings. The primary role of the LLM is to conduct generalized semantic reasoning activities, parsing unstructured terms on engineering drawings. The LLM acts not as an unrestrained controller but as an interpretative processor operating on prompts strictly structured by the subsequent layers.

3.2. Layer 2: Retrieval-Augmented Generation (RAG)

While an LLM provides generalized reasoning, specific operational knowledge is maintained via the RAG layer. The RAG component is built on a dataset of paired drawings and recipes and provides explicit contextual memory that the LLM accesses during query resolution. Furthermore, it is equipped with an integrated reasoning-chain component, based on generative code technologies and inspired by program-aided reasoning frameworks [25, 26], designed to analyze past recipe construction logic.

3.3. Layer 3: Knowledge Graph (KGraph)

To prevent hallucinations and guarantee deterministic decision bounds, the third layer introduces a structural Knowledge Graph. Rather than adopting a heavy, rigid OWL/RDF ontology, our current implementation uses a lightweight property graph structured via flat mapping configurations (e.g., `machine_map.yaml`). Drawing from Graph RAG approaches [27], the KGraph enforces awareness over the agent’s decision-making capabilities by acting as a deterministic look-up classification engine. For versioning, when new transformation mechanics are produced by the Scratch Generator, they are deterministically appended as new entity keys mapped to the reference workspace, maintaining a monotonic historical record of verified semantics without risking overwrites of established nodes.

4. Modular Multi-Agent Orchestration Workflow

Building upon the three-layered architecture, the operational execution is orchestrated through a modular multi-agent intelligent system. It is critical to clarify that, within the rigorous constraints of Industry 5.0, deploying a classical Multi-Agent System (MAS) characterized by emergent, decentralized negotiations (e.g., FIPA/ACL) introduces unacceptable degrees of non-determinism. Instead, we architected this system as a *Sequential Agentic Workflow* [28], a structured pipeline in which agents operate sequentially over a shared state rather than through decentralized interaction. In this MAS subset, highly specialized conversational agents (the modules) do not interact through open dialogue, but through a strictly typed shared state dictionary (`PipelineGraphState`) used to store and propagate intermediate data across modules. The workflow is governed by a centralized state-graph orchestrator (`LangGraph` [29]), and the agents collaborate through tightly controlled LLM dialectics [30]. This configuration sacrifices purist agent autonomy in favor of deterministic safeguards, ensuring that the agents’ reasoning capabilities remain contained within physical factory constraints. To detail the payload exchanges, the interaction logic is mapped via a sequence diagram in Figure 3, outlining the sequential pipeline and message passing between the five modules.

4.1. Module 1: Metadata Identifier

This initial module parses incoming manufacturing drawings to identify the macro-class and intrinsic characteristics based on embedded metadata. It normalizes unstructured data into classifiable objects that can be deterministically routed through the remaining layers. A critical vulnerability at this stage is LLM misclassification (e.g., classifying a clamping fixture as a welding torch). Such errors corrupt the state, passing an incorrect `macro_class` payload downstream.

4.2. Module 2: Batch Comparator

Operating sequentially after classification, this module verifies the requested production batch against existing repositories. It determines whether the newly required batch can be interpolated from similar

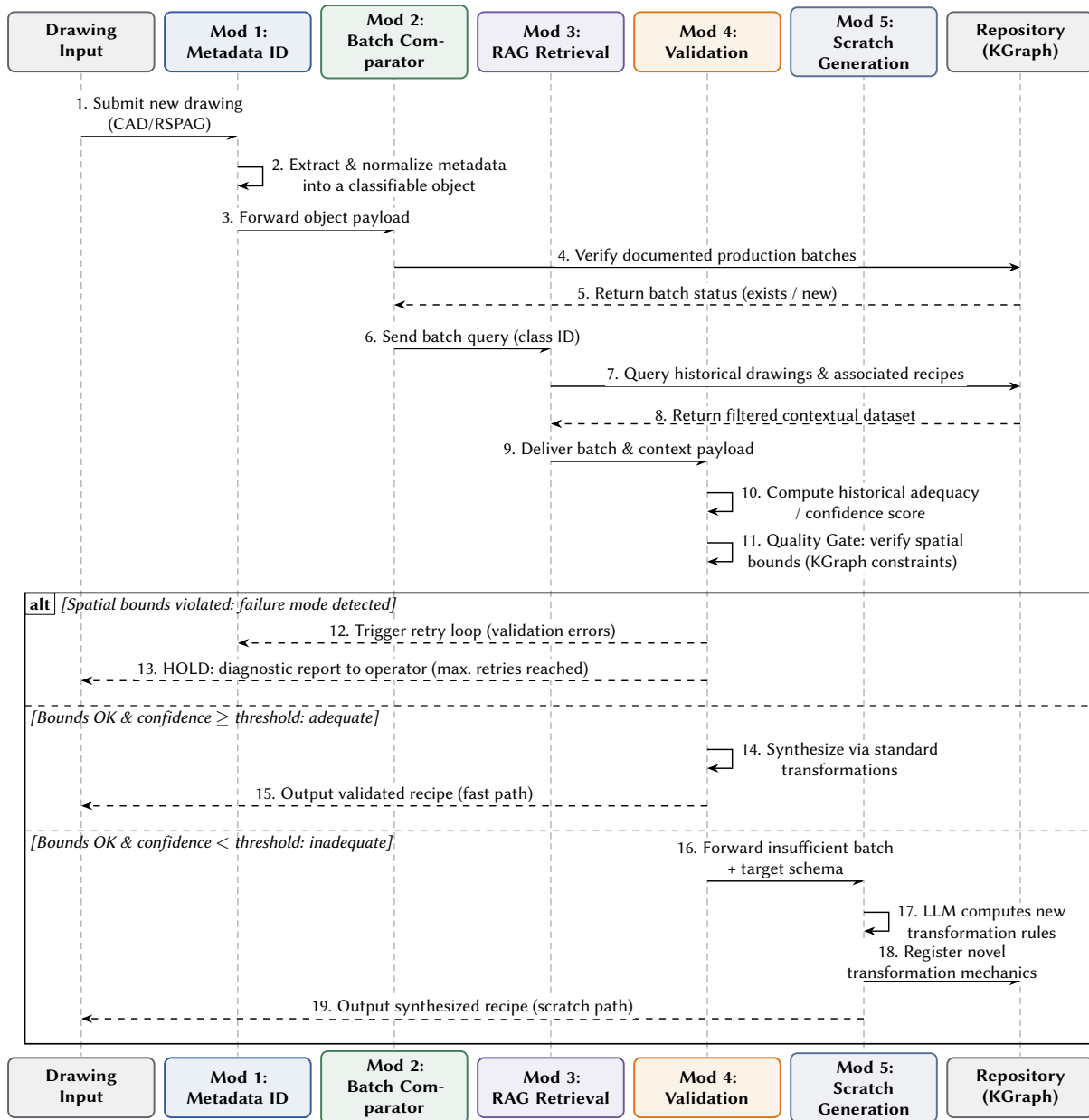


Figure 3: Sequence diagram illustrating the message exchange, payload routing, and decision bifurcation across the five agentic modules.

processes already documented in the private on-premise repository, or whether an entirely new recipe must be generated from scratch.

4.3. Module 3: RAG-Retrieval Module

This component is responsible for querying the knowledge repository. It retrieves all existing drawings and associated historical recipes in the private on-premise repository that match the geometric class identified by the Metadata Identifier. The output is a filtered subset of historical data exclusively relevant to the current engineering context.

4.4. Module 4: Validation and Adequacy Module

This strategic decision node evaluates the batch compiled by the Retrieval Module. It computes whether the accumulated historical data is robust enough to generate the new recipe via standard

transformations. Crucially, this module serves as a deterministic Quality Gate to contain upstream failures. If the RAG-Retrieval Module fetched incorrect context due to a Module 1 misclassification, the resulting spatial calculations will violate the hard physical reach limits defined in the KGraph. This module deterministically detects these anomalies, blocks execution, and forces a retry loop back to Module 1; if the violation persists after a bounded number of retries, the pipeline enters the HOLD state and issues a diagnostic report for operator intervention. In this way, neural hallucinations cannot propagate to the robotic controller. As highlighted in the decision flow diagram (Figure 4), if the data are deemed adequate and physically safe, the module synthesizes the novel recipe autonomously using verified prior transformation maps.

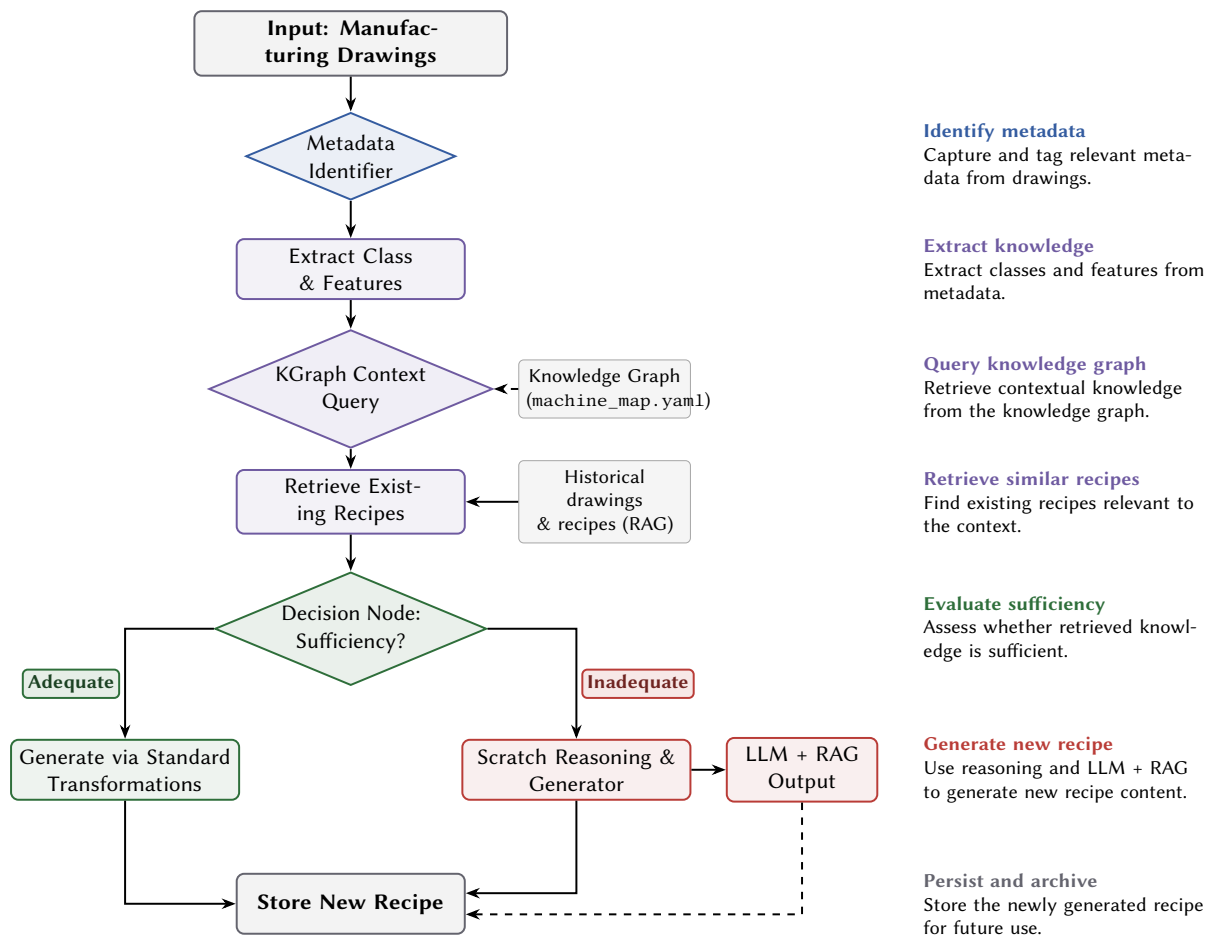


Figure 4: Pipeline decision flow outlining how input drawings are evaluated for historical adequacy versus scratch generation requirements.

4.5. Module 5: Scratch Reasoning and Generator Module

Should the Validation module determine that the pre-existing dataset is insufficient to formulate conservative standard transformations, the pipeline shifts execution to the Scratch Reasoning layer. Using the base LLM, this generative module computes a new recipe from scratch by evaluating a single novel drawing against the insufficient batch baseline. The central directive of this module is to compute and formally register new transformation mechanics, expanding the KGraph and RAG history for future pipeline queries.

5. Experimental Case Study: Assembly and Molding Automation

To validate the conceptual architecture, we deployed a qualitative pilot implementation focused on the extraction of distinct robotic components (abstracted here as “Small-Scale Assemblies” and “Large-Scale Molds” to maintain industrial confidentiality). It must be emphasized that this proof of concept aims strictly to validate the syntactic reliability and architectural feasibility of the neuro-symbolic pipeline versus traditional pure generative approaches. The pilot dataset comprised a micro-batch of four complex industrial archives incorporating multi-feature geometries (e.g., standard Tool Center Points and mold origins). The pipeline orchestrates an offline .NET microservice extractor and a Python-based LangGraph state machine [29] to safely generate ABB robotic outputs.

For clarity and relevance, the following case study focuses only on the subset of modules that were effectively activated during the pilot execution. While the full architecture comprises five modules, certain components (namely the Batch Comparator and the Scratch Reasoning and Generator Module) were not triggered in this specific scenario, as the available historical data were deemed sufficient for standard transformation. Consequently, the discussion below emphasizes the operational stages that were concretely exercised, without loss of generality with respect to the overall system design.

1. Data Extraction (Module 1). An input industrial archive from the pilot corpus, pseudonymized here as `SimArchive_Alpha.rspag`, is ingested. The deterministic extractor parses the proprietary format, outputting an agnostic JSON payload exposing physical traits without LLM hallucination risk. For example, a metadata snippet identifies an object via its physical features and naming semantics:

```
"part_name": "GRIPPER_ASSY",  
"aliases": ["PINZA", "GRIPPER"],  
"position_xyz": [120.0, 30.0, 410.0]
```

2. Semantic Anchoring (Module 3). A local LLM (e.g., Qwen2.5 [22]) evaluates the aliases against the predefined KGraph (structured via `machine_map.yaml`), resolving human naming ambiguities by identifying "PINZA" as the standard Tool Center Point (TCP).

3. Quality Gate Validation (Module 4). The pipeline routes the resolved context through the Quality Gate. Using deterministic numeric libraries, the relevant mathematical relationships (e.g., Euclidean distances, vector orientations) are calculated. If the spatial relationships exceed the expected physical work-cell boundaries, the graph interrupts the workflow, forcing a retry loop rather than allowing generation.

4. RAPID Output Generation (Module 4, adequate path). In our pilot run, the historical dataset was deemed sufficient (the “Adequate” path), hence the system generated the output via standard transformations without invoking the Scratch Generation pathway (Module 5), which is specified architecturally but was not activated in the reported pilot run. The generator produced executable recipe code in RAPID format, aligned with the deterministic bounds imposed by the validation module:

```
! Generated TCP Coordinates  
PERS tooldata tGripper:=[TRUE, [[120.0, 30.0, 410.0],  
[0.0, 0.0, 0.0, 1.0]], [1, [0, 0, 1], [1, 0, 0, 0], 0, 0, 0]];
```

This proof of concept shows that abstract engineering concepts can be safely translated into functional robot-ready targets. The validation threshold used in Module 4 was a strict, static Euclidean limit defined by manufacturing domain experts to reflect the absolute physical reach constraints of the robotic arm.

5.1. Quantitative Baseline Comparison: Pure LLM vs. Neuro-Symbolic Agent

To quantify the advantage of the proposed architecture, we conducted a comparative analysis against a baseline approach using a pure Large Language Model (Qwen2.5 3B operating in zero-shot mode). The baseline model was prompted to directly translate the extracted JSON metadata into RAPID code without the scaffolding of the Knowledge Graph or the Validation Quality Gate.

The evaluation focused on three primary metrics: processing latency, syntactic validity of the RAPID code, and deterministic geometric safety. Latency is measured as the end-to-end inference time. Syntactic validity refers to compliance with RAPID syntax, while geometric safety is defined as the absence of coordinates violating predefined workspace constraints, i.e., the absence of out-of-bounds or collision-risk trajectories, such as negative Z-axis coordinates.

While the pure LLM baseline showed lower latency due to its single-pass inference mechanism (average 2.15 seconds versus the pipeline’s 3.82 seconds), it exhibited critical vulnerabilities. The unconstrained LLM failed to reliably format the proprietary `wobjdata` and `robtarg` structures. More critically, the baseline generated spatially hallucinated coordinates that violated physical constraints ($Z < 0$), rendering the output undeployable. In contrast, the neuro-symbolic pipeline achieved 100% geometric safety within the pilot set. The deterministic constraints enforced by the Quality Gate (Module 4) intercepted probabilistically hallucinated trajectories, ensuring that only physically validated and syntactically valid robotic routines were synthesized. Although limited to the pilot set, this comparison indicates that standalone generative models lack the safety guarantees required in Industry 5.0 settings, whereas the proposed framework enforces them by construction.

6. Results and Discussion

The novelty of this work lies in the explicit integration of a deterministic Quality Gate within a multi-layer neuro-symbolic architecture, enabling the enforcement of physical constraints over generative outputs in industrial settings.

The pilot implementation provides preliminary evidence that the proposed neuro-symbolic architecture can improve the reliability of LLM-assisted recipe generation in safety-sensitive manufacturing contexts. Although the experimental setting is intentionally limited to a micro-batch of four confidential industrial archives, the comparison with a pure LLM baseline highlights a clear trade-off between execution speed and operational safety.

Table 1 summarizes the main observed outcomes. The pure LLM baseline achieved lower latency because it relied on a single-pass generation process. However, this advantage was obtained at the cost of reduced reliability: the model produced syntactic inconsistencies in RAPID structures and generated at least one spatially invalid trajectory, including coordinates outside the admissible workspace. By contrast, the proposed neuro-symbolic pipeline introduced additional processing overhead due to metadata extraction, KGraph grounding, RAG-based contextualization, and Quality Gate validation. Nevertheless, this overhead remained modest and was associated with complete syntactic validity and geometric safety in the pilot evaluation.

Table 1

Summary of pilot results comparing a pure LLM baseline with the proposed neuro-symbolic pipeline.

Evaluation Criterion	Pure LLM Baseline	Neuro-Symbolic Pipeline
Average processing latency	2.15 s	3.82 s
Syntactic RAPID validity	Not consistently satisfied	Fully satisfied in the pilot set
Geometric safety	Violations observed	100% safe in the pilot set
Out-of-bounds trajectories	Present, including $Z < 0$ cases	Not observed
Use of historical context	None	RAG-supported retrieval
Use of explicit constraints	None	KGraph and Quality Gate validation
Privacy-preserving execution	Local model possible	Local and structurally constrained
Human intervention policy	Not explicit	Triggered under uncertainty or violation

These results suggest that the central contribution of the proposed framework is not merely improved code generation, but controlled generation. In conventional software synthesis, an incorrect output can often be detected through compilation errors or unit tests. In industrial manufacturing, however,

syntactically valid code may still be physically unsafe. This distinction motivates the introduction of the Quality Gate as a deterministic boundary between probabilistic reasoning and executable production logic.

The observed latency increase from 2.15 to 3.82 seconds appears acceptable in the considered scenario, since recipe generation is not a hard real-time control task but an upstream planning activity. More importantly, the additional time is used to enforce validation steps that directly reduce deployment risk. In this sense, the proposed architecture privileges verifiability over raw generation speed, which is consistent with the requirements of trustworthy AI in Industry 5.0.

A further relevant aspect concerns explainability. The sequential organization of the pipeline makes it possible to identify where a failure occurs: metadata misclassification, insufficient historical context, violation of spatial constraints, or need for scratch generation. This modularity is beneficial for human-in-the-loop supervision, since operators can inspect intermediate states rather than receiving only an opaque generated output.

The main limitation of the current results is the restricted size and confidentiality of the pilot dataset. The evaluation should therefore be interpreted as an architectural feasibility study rather than as a statistically generalizable benchmark. Future experiments will require a larger and more diverse set of CAD/RSPAG inputs, systematic stress testing of misclassification scenarios, and quantitative analysis of the Quality Gate thresholds under different machine configurations. Nevertheless, the pilot evidence supports the core hypothesis of this work: LLMs can be useful in industrial recipe synthesis when their generative flexibility is embedded within retrieval mechanisms, symbolic constraints, and deterministic validation procedures.

7. Conclusion and Future Work

The integration of Large Language Models into industrial recipe generation offers a promising opportunity to bridge engineering abstraction and executable manufacturing knowledge. However, the results discussed in this paper indicate that pure generative approaches remain insufficient for factory environments, where syntactic correctness alone does not guarantee physical feasibility or operational safety. In this work, we proposed a hybrid neuro-symbolic framework for autonomous recipe synthesis, combining local LLM reasoning, retrieval-augmented contextualization, and Knowledge Graph-based constraint enforcement within a modular five-stage agentic workflow.

The current study is also subject to several limitations. The evaluation is restricted to a small and confidential dataset, and the system has not yet been validated under large-scale industrial variability. Furthermore, the Scratch Generation pathway has not been empirically assessed in this work.

The main contribution of the paper is the definition of an architecture in which generative flexibility is deliberately bounded by deterministic verification. Rather than treating the LLM as an autonomous controller, the proposed framework assigns it a constrained semantic role, while geometric reasoning, historical adequacy assessment, and execution validity remain governed by symbolic and traceable mechanisms. The preliminary case study suggests that this separation is essential for trustworthy industrial AI: the additional latency introduced by validation is compensated by stronger guarantees in terms of syntactic validity, geometric safety, and explainability.

Future work will proceed along three complementary development directions.

First, the immediate priority is the direct implementation and stress testing of the proposed architecture on real-world industrial data. This step will involve large-scale validation on heterogeneous CAD/RSPAG archives, different robot configurations, diverse machine classes, and production recipes with varying degrees of geometric and semantic complexity. Particular attention will be devoted to failure-mode analysis, including metadata misclassification, retrieval mismatch, borderline geometric violations, and insufficient historical coverage. The Validation and Adequacy Module will be empirically calibrated using real operational thresholds, safety margins, and expert-defined constraints, enabling a rigorous comparison with both human programming workflows and pure LLM baselines.

Second, the architecture will be extended toward closed-loop simulation and digital-twin-based self-

verification. Digital twins are increasingly recognized as a central mechanism for connecting physical manufacturing systems with virtual simulation, monitoring, and optimization environments [13, 31]. Recent research has also explored the integration of LLMs with digital twins for more interpretable and adaptive industrial reasoning [32, 33]. In this direction, the generated recipe will not be considered complete after symbolic validation alone, but will be executed in a simulation environment before deployment. Such a loop would enable collision checking, reachability analysis, cycle-time estimation, tool-path inspection, and process feasibility testing. The resulting feedback could then be reinjected into the RAG and KGraph layers, allowing the system to refine candidate recipes before they reach the physical production line. This would transform the proposed framework from a validated generator into a simulation-aware cyber-physical reasoning system.

Third, future research will investigate continual industrial memory through adaptive Knowledge Graph and RAG evolution. Instead of treating the knowledge base as a static repository, successful recipes, rejected generations, operator corrections, and simulation failures can be progressively encoded as versioned industrial experience. This direction is aligned with recent work on industrial Knowledge Graphs and their role in smart manufacturing [6, 7]. A long-term objective is to associate transformation rules, machine constraints, and recipe families with confidence scores, temporal validity, and provenance metadata. In this way, the system could gradually capture tacit engineering knowledge, distinguish stable manufacturing principles from context-dependent heuristics, and support safer forms of continual adaptation. Such an extension would make the architecture not only a recipe synthesis pipeline, but also a corporate memory layer for trustworthy and evolving industrial intelligence.

Overall, the proposed framework represents an initial step toward reliable neuro-symbolic agents for Industry 5.0 manufacturing. Its broader ambition is to enable generative AI systems that are not merely capable of producing industrial code, but are also able to justify, verify, simulate, and progressively improve their outputs under explicit safety and traceability constraints.

Acknowledgments

The authors would like to express their gratitude to EssilorLuxottica for supporting this research and fostering a fruitful collaboration between academia and industry. Alessandro Venaruzzo gratefully acknowledges the company for the financial support provided within the PNRR program “Tecniche di Machine Learning per la generazione automatica di codice di controllo per macchinari di automazione industriale”. This support has been instrumental in enabling the development of the research presented in this work.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT in order to: Grammar and spelling check, paraphrase and reword. After using these tools/services, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication’s content.

References

- [1] P. K. R. Maddikunta, Q.-V. Pham, B. Prabadevi, N. Deepa, K. Dev, T. R. Gadekallu, R. Ruby, M. Liyanage, Industry 5.0: A survey on enabling technologies and potential applications, *Journal of Industrial Information Integration* 26 (2022) 100257.
- [2] S. Vemprala, R. Bonatti, A. Bucker, A. Kapoor, ChatGPT for Robotics: Design Principles and Model Abilities, Technical Report, Microsoft Autonomous Systems and Robotics Research, 2023.
- [3] Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, J. Sen, Y. Zhou, et al., The rise and potential of large language model based agents: A survey, *arXiv preprint arXiv:2309.07864* (2023).

- [4] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, D. Kiela, Retrieval-augmented generation for knowledge-intensive nlp tasks, in: *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020, pp. 9459–9474.
- [5] A. Hogan, E. Blomqvist, M. Cochez, C. d’Amato, G. D. Melo, C. Gutierrez, S. Kiriane, J. E. L. Gayo, R. Navigli, S. Neumaier, et al., Knowledge graphs, *ACM Computing Surveys (CSUR)* 54 (2021) 1–37.
- [6] Y. Shang, Y. Ren, H. Peng, Y. Wang, G. Wang, Z. Li, Y. Yang, Y. Li, A perspective survey on industrial knowledge graphs: Recent advances, open challenges, and future directions, in: *Proceedings of the International Conference on Machine Learning and Cybernetics (ICMLC)*, 2023.
- [7] C. Zhou, et al., Making knowledge graphs work for smart manufacturing: Research topics, applications, and challenges, *Robotics and Computer-Integrated Manufacturing* 89 (2024) 102790.
- [8] A. d’Avila Garcez, L. C. Lamb, Neurosymbolic ai: The 3rd wave, *Artificial Intelligence Review* 56 (2023) 12387–12406.
- [9] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, D. Mané, Concrete problems in ai safety, *arXiv preprint arXiv:1606.06565* (2016).
- [10] G. Dulac-Arnold, S. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, T. Hester, Challenges of real-world reinforcement learning: Definitions, benchmarks and analysis, *Machine Learning* 110 (2021) 2419–2468.
- [11] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, J. Zhang, Edge intelligence: Paving the last mile of artificial intelligence with edge computing, *Proceedings of the IEEE* 107 (2019) 1738–1762.
- [12] Z. Yu, Z. Wang, Y. Li, H. You, R. Gao, X. Zhou, S. R. Bommur, Y. Zhao, Y. C. Lin, Edge-llm: Enabling efficient large language model adaptation on edge devices via layerwise unified compression and adaptive layer tuning and voting, in: *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*, 2024.
- [13] F. Tao, M. Zhang, A. Y. C. Nee, *Digital Twin Driven Smart Manufacturing*, Academic Press, 2019.
- [14] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, et al., Do as i can, not as i say: Grounding language in robotic affordances, in: *Conference on Robot Learning (CoRL)*, PMLR, 2022.
- [15] T. R. Sumers, S. Yao, K. Narasimhan, T. L. Griffiths, Cognitive architectures for language agents, *arXiv preprint arXiv:2309.02427* (2023).
- [16] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al., Evaluating large language models trained on code, *arXiv preprint arXiv:2107.03374* (2021).
- [17] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, et al., Competition-level code generation with alphacode, *Science* 378 (2022) 1092–1097.
- [18] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, C. Xiong, Codegen: An open large language model for code with multi-turn program synthesis, in: *International Conference on Learning Representations*, 2023.
- [19] Y. Wang, H. Le, A. D. Gotmare, N. D. Q. Bui, J. Li, S. C. H. Hoi, Codet5+: Open code large language models for code understanding and generation, in: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 1069–1088.
- [20] J. Jiang, F. Wang, J. Shen, S. Kim, S. Kim, A survey on large language models for code generation, *arXiv preprint arXiv:2406.00515* (2024).
- [21] J. Liang, W. Huang, F. Xia, P. Peng, K. Hausman, B. Ichter, P. Florence, A. Zeng, Code as policies: Language model programs for embodied control, in: *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023, pp. 9493–9500.
- [22] A. Yang, B. Baige, B. Biao, et al., Qwen2 technical report, *arXiv preprint arXiv:2407.10671* (2024).
- [23] Ollama Team, Ollama: Get up and running with large language models locally, <https://github.com/ollama/ollama>, 2024. Accessed: 2026-05-04.
- [24] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y.-I. Bang, A. Madotto, P. Fung, Survey of

- hallucination in natural language generation, *ACM Computing Surveys* 55 (2023) 1–38.
- [25] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, G. Neubig, Pal: Program-aided language models, *arXiv preprint arXiv:2211.10435* (2023).
 - [26] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafraan, K. Narasimhan, Y. Cao, React: Synergizing reasoning and acting in language models, *arXiv preprint arXiv:2210.03629* (2023).
 - [27] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, J. Larson, From local to global: A graph rag approach to query-focused summarization, *arXiv preprint arXiv:2404.16130* (2024).
 - [28] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, et al., A survey on large language model based autonomous agents, *Frontiers of Computer Science* 18 (2024) 1–26.
 - [29] LangChain AI, Langgraph: Build resilient language agents as graphs, <https://github.com/langchain-ai/langgraph>, 2024. Accessed: 2026-05-04.
 - [30] Z. Mandi, S. Shirjian, S. Haldar, et al., Roco: Dialectic multi-robot collaboration with large language models, *arXiv preprint arXiv:2307.04738* (2023).
 - [31] D. Jones, C. Snider, A. Nassehi, J. Yon, B. Hicks, Characterising the digital twin: A systematic literature review, *CIRP Journal of Manufacturing Science and Technology* 29 (2020) 36–52.
 - [32] Y. Xia, et al., An architecture for integrating large language models with digital twins and automation systems, *arXiv preprint arXiv:2501.13179* (2025).
 - [33] L. Yang, S. Luo, X. Cheng, L. Yu, Leveraging large language models for enhanced digital twin modeling: Trends, methods, and challenges, *arXiv preprint arXiv:2503.02167* (2025).