Automatisierte SPS-Programmierung: Generierung von qualitativ hochwertigem Steuerungscode nach IEC 61131-3

Automating PLC Programming: Generating high-quality control code compliant with IEC 61131-3

R. Kimmel, Universität Stuttgart, Stuttgart;
Dr. **B. Wiesmayr**, Johannes Kepler Universität Linz, Linz;
Univ.-Prof. Dr. **A. Wortmann**, Universität Stuttgart, Stuttgart;

Kurzfassung

Der Beitrag präsentiert einen Entwurf für die autonome Generierung von Steuerungscode nach IEC 61131-3 als Strukturierter Text. Steuerungscode nach IEC 61131-3 findet in Anlagen und Maschinen Anwendung, deren Zweck in der Automatisierung von Prozessen und Abläufen besteht. In der Regel werden für diese Aufgabe Domänenexperten eingesetzt, um eine aufgabenbezogene Logik zu entwickeln und zu implementieren. Der vorliegende Entwicklungsprozess ist als komplex, aufwendig, fehleranfällig und folglich kostenintensiv zu bewerten. Das von uns entwickelte Konzept kann als autonomer, auf künstlicher Intelligenz basierender Programmier-Assistent oder im vollautonomen Betrieb verwendet werden, und soll den vorliegenden Entwicklungsprozess signifikant vereinfachen, verkürzen, sowie robuster und damit insgesamt günstiger machen. Im Wesentlichen basiert das Konzept auf einem Large-Language-Model-Agenten, der Anforderungen an eine Automatisierungslösung in konkreten Steuerungscode umwandelt. Dafür fließt zuvor modelliertes Wissen, unter anderem über Prozesse und Programmierung von Steuerungscode, via „Retrieval-Augmented Generation“ in den Planungs- und Umsetzungsprozess des Agenten ein. Der entwickelte Steuerungscode wird von dem Agenten eigenständig mittels statischer Analyse (Compiler/Linter) und durch Testszenarien in einer Simulation überprüft und iterativ verbessert.

Abstract

This paper presents a concept for the autonomous generation of control code in accordance with IEC 61131-3 Structured Text. Such control code is used in systems and machines whose purpose is to automate processes and sequences. Usually, domain experts are tasked to develop and implement related logic according to given requirements. This engineering process is complex, time-consuming, error-prone and therefore cost-intensive. Our concept can be used as an autonomous programming assistant based on artificial intelligence, or operate fully autonomously, and is intended to significantly simplify and shorten the existing engineering process,

make it more robust, and therefore overall cheaper. At its core, the concept is based on a Large Language Model agent that converts the requirements of an automation solution into concrete control code. To achieve high-quality control code, previously modeled knowledge (e.g., about processes or control code programming) flows into the agent's planning and implementation process via “Retrieval-Augmented Generation”. The developed control code is autonomously checked and iteratively improved by the agent via static analysis (compiler/linter) and test scenarios in a simulation.

1. Einleitung

Automatisierungssysteme werden auch als cyberphysische Systeme (CPSs) bezeichnet und umfassen Komponenten aus den Bereichen Mechanik, Elektrotechnik sowie Steuerung und Regelungstechnik, die in einem integrierten Ansatz miteinander kombiniert werden [1]. Für deren Entwicklung ist ein aufwendiges Zusammenspiel verschiedener Disziplinen erforderlich [2]. Dies geschieht hauptsächlich durch den Austausch von Dokumenten [1]. Die Programmierung solcher CPSs erfolgt in der Regel auf Basis von Speicherprogrammierbaren Steuerungen (SPS), welche häufig gemäß dem IEC 61131-3-Standard [3] in Strukturiertem Text programmiert werden. Meistens entwickeln Ingenieure und nicht Softwareentwickler SPS-Programme [4]. Dies ist darauf zurückzuführen, dass für die Programmierung vertieftes Wissen bezüglich der Domäne und der zu automatisierenden Prozesse notwendig ist [5]. In komplexen Automatisierungssystemen sind die SPS-Programme ebenfalls sehr groß. Die Entwicklung und Wartung erweist sich daher häufig als zeitintensive und fehleranfällige Aufgabe [4]. Einer höheren Qualität der Automatisierungssoftware, welche die Entwicklungsarbeiten vereinfachen könnte, stehen häufig Herausforderungen wie Fachkräftemangel und Personalkosten entgegen. Automatisierungslösungen für die adäquate Messung von Programmkomplexität [5, 6] und zur Verbesserung von Codequalität [6, 7] können daher hilfreich sein. Der Effekt potenziert sich in Fällen, in denen variantenreiche Steuerungsprogramme verwaltet werden müssen, zum Beispiel aufgrund individueller Anpassungen oder einer veränderten Liefersituation und dadurch bedingter Umplanung der bestehenden Automatisierungslösungen. Auch Varianten können systematisch erfasst und umgesetzt werden [8]. Eine fortschreitende Automatisierung bedingt demnach einen erhöhten Entwicklungsaufwand, der wiederum die erhöhte Verfügbarkeit qualifizierter und erfahrener Fachkräfte erfordert. Eine (teilweise) Automatisierung des zugehörigen Softwareentwicklungsprozesses kann zur Lösung dieser Probleme beitragen und insgesamt eine Verbesserung der Softwarequalität erreichen. Dazu soll im Folgenden der Einsatz von Künstlicher Intelligenz (KI) in Form von großen Sprachmodellen (Large Language Models, LLMs) vorgestellt werden.

2. Stand der Forschung und Entwicklung

Verschiedene Arten der Generierung von SPS-Programmen aus abstrakten Beschreibungen sind möglich. Eine etablierte Variante ist die Codegenerierung [9] aus Softwaremodellen [10] oder textuellen Spezifikationsdokumenten [11]. Auch Transformationen zwischen verschiedenen Artefakten sind denkbar. Das Wissen über Prozesse, Systemarchitekturen bzw. Anforderungen wird als Modelle oder Regeln abgelegt und bildet die Grundlage für eine Codegenerierung [11]. Im Gegensatz zu diesen Methoden können LLMs natürlichsprachliche Eingaben verarbeiten und Kontext interpretieren.

Large Language Models

LLMs haben sich seit der Einführung der Transformer-Architektur [12] rasant weiterentwickelt und sind grundsätzlich auch für die Generierung von Steuerungscode geeignet [13, 14]. Modelle wie GPT-4 [15] und Claude 3 [16], aber auch Open-Source-Modelle wie Llama 4 [17] und DeepSeek v3 [18], zeigen bemerkenswerte Fähigkeiten in der Verarbeitung und Generierung von Text [19] sowie im Verständnis komplexer Zusammenhänge. Im technischen Kontext sind besonders ihre Fähigkeiten zur Mustererkennung und zum Verständnis kontextabhängiger Informationen relevant. LLMs dominieren seither das Feld des Natural Language Processing (NLP), werden aber zunehmend auch in anderen Bereichen, wie z.B. der Mathematik, eingesetzt [20]. In der Regel werden LLMs mit großen Textmengen, z.B. aus dem Internet, trainiert. Da dieser Prozess sehr energieintensiv und damit teuer ist, werden für Endanwendungen häufig vortrainierte Modelle eingesetzt. Diese können aber über diverse Strategien für spezifische Anwendungen angepasst werden. Dazu wird im Folgenden drei wichtige Aspekte betrachtet.

Prompt Engineering ermöglicht die Steuerung des Verhaltens von LLMs durch strukturierte Eingabeaufforderungen. Der Prozess umfasst unter anderem die Definition der Rolle des Modells (z.B. technischer Analyst), spezifische Handlungsanweisungen und gewünschte Ausgabeformate [21, 22]. So können typische Fehler durch entsprechende Prompts vermieden werden [13]. Durch die Integration von Beispielen (Few-Shot Learning) [23] und der Zerlegung komplexer Aufgaben in logische Teilschritte (Chain-of-Thought Prompting) [24], kann die Verlässlichkeit der Ausgabe systematisch optimiert werden. Ein kontinuierliches Testen und Verfeinern der Prompts gewährleistet die für technische Anwendungen notwendige Präzision.

Retrieval-Augmented Generation (RAG) erweitert die Fähigkeiten von LLMs durch die dynamische Integration von externem Wissen [25]. Der Prozess beginnt z.B. mit der Aufbereitung domänenspezifischer Dokumente in Vektorrepräsentationen, die in spezialisierten Datenbanken gespeichert werden. Im Falle einer Anfrage werden relevante Informationen durch Ähnlichkeitssuche extrahiert und zusammen mit der eigentlichen Aufgabenstellung an das LLM übergeben.

Dieser Ansatz kombiniert die Flexibilität von LLMs mit der Genauigkeit dokumentierter Fachexpertise und reduziert gleichzeitig das Risiko von Halluzinationen, da das Modell auf verifizierte Informationen zugreift. Auch andere Arten der Informationsbereitstellung sind hierbei denkbar und gegebenenfalls sinnvoll.

Fine-Tuning passt vortrainierte LLMs durch gezieltes Nachtraining an spezifische Anwendungen an. Dabei wird das Modell mit einem domänenspezifischen Datensatz nachtrainiert und lernt so die Besonderheiten der Fachsprache und typische Aufgabenmuster. Moderne Verfahren wie Parameter-Effizientes Fine-Tuning (PEFT) [26] oder Low-Rank Adaptation (LoRA) [27] ermöglichen diese ressourcenschonende Anpassung, die nur wenige Modellparameter verändert. Der Prozess benötigt typischerweise wesentlich weniger Beispiele als das initiale Training und führt zu einem spezialisierten Modell, das Domänenwissen beherrscht und genauere Ergebnisse für die Zielanwendung liefern kann.

Testen und Validieren von Software

Die korrekte Funktionalität ist ein wesentlicher Aspekt von Softwarequalität und kann mit unterschiedlichen Methoden sichergestellt werden.

Softwaretests sind ein zentraler Bestandteil der Qualitätssicherung in der Softwareentwicklung [28]. Insbesondere in der industriellen Automatisierung stellen die Verifikation und Validierung von Steuerungssoftware eine wichtige Herausforderung dar, da Fehler zu erheblichen Produktionsausfällen oder Sicherheitsrisiken führen können. Traditionelle Testmethoden umfassen Unit-Tests, Integrationstests sowie Systemtests unter realen Bedingungen [28]. Diese Ansätze sind jedoch oft zeit- und kostenintensiv, insbesondere wenn physische Anlagen oder Prototypen benötigt werden. Darüber hinaus steigt mit zunehmender Komplexität der Systeme der Bedarf an frühzeitigen, automatisierten Testverfahren, die eine systematische Verifikation der Steuerungslogik ermöglichen, noch bevor die reale Hardware verfügbar ist.

Virtuelle Inbetriebnahme (VIBN) ist ein zunehmend etablierter Ansatz zur Effizienzsteigerung beim Testen von Steuerungssoftware [29]. Dabei wird ein digitales Modell der realen Maschine oder Anlage erstellt und zusammen mit der Steuerungssoftware simuliert. Dies ermöglicht eine umfassende Testabdeckung unter realitätsnahen Bedingungen ohne physikalische Ressourcen zu beanspruchen. Durch die Kopplung von Steuerungsprogrammen mit virtuellen Anlagenmodellen können Fehler, die durch die Interaktion mit der Umwelt entstehen, frühzeitig erkannt und behoben werden, was Entwicklungszeiten verkürzt und Kosten senkt. Gleichzeitig eröffnet VIBN neue Möglichkeiten für Continuous Integration und automatisiertes Testen, was insbesondere in agilen Entwicklungsprozessen von hoher Relevanz ist [29, 30].

3. Problemstellung und Anforderungen

Obwohl LLMs grundsätzlich beachtliche Fähigkeiten in der Erstellung von Programmcode aufweisen, ist die Generierung qualitativ hochwertiger SPS-Programme allein durch Sprachmodelle nicht ausreichend gewährleistet [13]. Die Generierung von Steuerungscode durch LLMs ist ein Prozess, der von verschiedenen Faktoren beeinflusst werden kann. Ein wesentlicher Aspekt ist dabei die Verfügbarkeit von geeignetem Wissen zum richtigen Zeitpunkt, das die Grundlage für eine hochwertige Generierung bildet. Beispielsweise kann die Einbindung von Bibliotheken über RAG Codeduplikate reduzieren [31]. Ein weiterer entscheidender Faktor ist die Validierung des erzeugten Codes, die sicherstellt, dass die erstellten Lösungen den gewünschten Anforderungen entsprechen oder nötigenfalls angepasst werden können. Die vorliegende Forschungsfrage lautet folglich (RQ): *Wie muss eine Softwarearchitektur unter Einbindung von LLMs gestaltet sein, damit sie in der Lage ist, basierend auf verschiedenen Anforderungen qualitativ hochwertigen Steuerungscode zu generieren?* Um diese Forschungsfrage zu beantworten werden vier als relevant identifizierte Teilfragen aufgeworfen und nachfolgend ausgeführt.

Zu den Kernaufgaben einer KI-gestützten Entwicklung zählt die Orchestrierung des Software-Entwicklungsprozesses. Das LLM muss daher die Fähigkeit besitzen, in der Art und Weise eines Menschen mit den diversen Teilaspekten des Entwicklungsprozesses zu interagieren. Die Aufgabe erfordert die Fähigkeit, Steuerungsprogramme zu generieren und diese in einem SPS-Projekt adäquat zu organisieren. Darüber hinaus umfasst sie die Kompilierung und Konfiguration solcher Projekte, also die Interaktion mit einer SPS-Software-Entwicklungsumgebung. Auch potenziell weitere relevante Softwaretools, die in der Entwicklung zum Einsatz kommen könnten, sollten durch das LLM verwendbar sein. Damit ergibt sich die Teilfragestellung: *Wie kann ein LLM dazu befähigt werden, mit dessen Umwelt, konkret einer Softwareentwicklungs-Landschaft, zu interagieren? (RQ1)*

In Analogie zu einem menschlichen Entwickler soll das LLM über eine Wissensbasis verfügen, die es bei der Erstellung von Programmcode verwendet. Zu den erforderlichen Kompetenzen zählen beispielsweise generelles Wissen über die Programmierung von Steuerungen, bewährte Verfahren, unternehmensspezifische Richtlinien, Kenntnisse der Maschinenparameter und Anschlüsse sowie Prozesswissen. Es sei darauf hingewiesen, dass einige Aspekte, wie beispielsweise die korrekte Syntax von SPS-Code nach IEC 61131-3, einem fähigen LLM durch das Vortrainieren inhärent sind. Andererseits gibt es Aspekte, die spezifische Maschinen, Anschlüsse und Firmenpolicen betreffen und daher in einem generischen Sprachmodell nicht abgebildet sind. Daher ist es erforderlich, dass dem LLM Wissen zur Generierung von Code-

Elementen adäquat bereitgestellt wird. In Abbildung 1 wird das zugrundeliegende Problem sowie die sich daraus ergebenden Fragen veranschaulicht. Einerseits ist die unidirektionale Fähigkeit des Erlernens, also dem Speichern von Wissen, zu betrachten. Dem steht die bidirektionale Fähigkeit gegenüber, erlerntes Wissen korrekt unter Zuhilfenahme von Kontextinformationen wieder dem Prozess oder Nutzer bereitzustellen. Die vorliegende Fragestellung lässt sich konkretisieren: *Wie sollte eine Wissensbasis für ein LLM modelliert werden, damit dieses hochqualitative SPS-Programme generieren kann, und welche Mechanismen sind notwendig, um Wissen in diesen Speicher aufzunehmen und korrekt aus ihm abzurufen? (RQ2)*

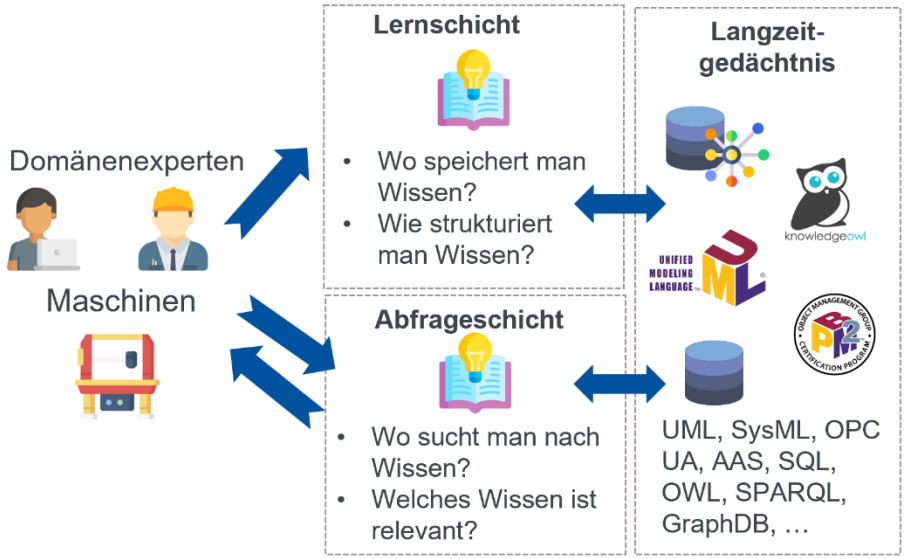

Abbildung 1: Persistente und einem LLM zugängliche Wissensspeicherung für KI-gestützte Codegenerierung

Praktisch alle Vorgehensmodelle für die Softwareentwicklung beinhaltet Phasen, die der Evaluierung des erstellten Codes dienen. Im Anschluss an jede Evaluierungsphase folgt in der Regel ein iterativer Ablauf um etwaige Fehler zu beheben und Optimierungen vorzunehmen. Diese sollten auch in die LLM-basierte Entwicklung zu integrieren um die Korrektheit der generierten Programme zu erhöhen. Dabei sollte der KI analog zum menschlichen Entwickler die Fähigkeit zur eigenständigen Verbesserung des erstellten Codes verliehen werden. Da der erstellte Steuerungscode cyberphysische Systeme ansteuert, reichen klassische Unit-Tests nicht aus, um die Semantik, also die Funktionsweise des Codes zu überprüfen. Dabei können bestehende Konzepte der VIBN verwendet werden. Es ergeben sich daher folgende Forschungsfragen für diesen Aspekt: *Wie kann die Funktionsweise von generiertem Steuerungscode inklusive seines physischen Aspektes validiert werden? (RQ3)* bzw. *Wie kann ein für LLMs verständliches Feedback aus dem Evaluierungsprozess gewonnen werden? (RQ4)*

4. Konzept

Abbildung 2 bietet einen Überblick über den gesamten Prozessablauf. Dieser startet mit einem initialen Prompt des Nutzers (links), welcher Anforderungen an das Programm enthält.

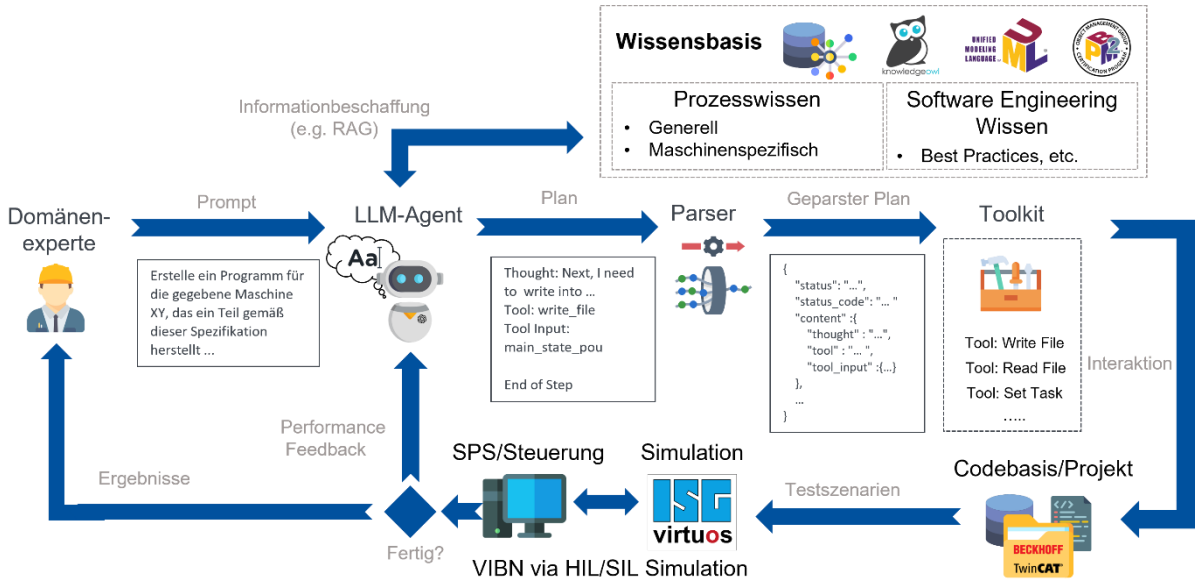

Abbildung 2: Ablauf des iterativen Code Generationsprozesses mit dem LLM Agenten im Zentrum als Prozessorchestrator sowie der Wissensbasis und Programmier- und Testumgebung

Agent (RQ1): Im Zentrum des vorgeschlagenen Konzeptes steht ein LLM-Agent. Dieser dient als zentraler Prozessorchestrator und ermöglicht dem LLM, mit der Umwelt zu interagieren. Dabei handelt es sich um ein Sprachmodell, das durch Prompt Engineering oder Fine-Tuning zu einem agentischen Verhalten gebracht wird. Der Agent handelt schrittweise, dazu antwortet das LLM immer nach einem bestimmten Schema, welches einen Gedanken und einen Funktionsaufruf enthält (im Zentrum von Abbildung 2 dargestellt). Der Agent erstellt einen Plan bzw. einen Schritt eines Plans in Textform. Der Text muss immer dem gegebenen Format folgen, damit der Parser die relevanten Felder des Schemas extrahieren kann. Der Agent verfügt über eine definierte Menge an Werkzeugen bzw. Funktionen, wie z.B. dem Lesen von Code. Der Agent weiß also, welche Werkzeuge zur Verfügung stehen und wählt dann für den aktuellen Schritt ein passendes Werkzeug aus, vom dem er sich verhofft, der Lösung näher zu kommen. Jeder Funktionsaufruf liefert Feedback an den Agenten, das Informationen über seine Umwelt enthält, z.B. den Inhalt einer Codeeinheit. Damit der Werkzeugkasten des Agenten für den gegeben Anwendungsfall der SPS-Programmierung funktioniert, werden Schnittstellen einer Entwicklungsumgebung, hier TwinCat [32], genutzt.

Wissensbasis (RQ2): Die Wissensbasis (oberer Teil von Abbildung 2) ist für den Agenten jederzeit über RAG-ähnliche Zugriffe verfügbar. Zur Modellierung einer geeigneten Wissensbasis für LLMs, die SPS-Programme generieren sollen, ist ein hybrider Ansatz sinnvoll, der strukturierte und unstrukturierte Wissensquellen kombiniert. Eine vielversprechende Umsetzung ist die Integration eines intelligenten Agenten, der Wissen automatisch extrahiert, klassifiziert und in einer geeigneten Speicherstruktur ablegt. Dazu nutzt der Agent Vektordatenbanken (z.B. Chroma [33]) für semantische Ähnlichkeitssuchen auf unstrukturierten Textdaten sowie relationale SQL-Datenbanken (z.B. PostgreSQL [34]) für hochstrukturierte Informationen wie Maschinenparameter oder Verbindungslisten. Darüber hinaus kann der Agent graphbasierte Wissensrepräsentationen (z.B. Neo4j [35] oder Ontologien in OWL [36]) aufbauen, um Beziehungen zwischen einzelnen Wissenselementen explizit abzubilden. Für die Modellierung komplexerer Strukturen wie Systemarchitekturen oder Prozessabläufe können standardisierte Metamodelle wie SysML [37], OPC UA [38] oder AAS [39] eingebunden werden. Der Agent übernimmt nicht nur das Einpflegen von Wissen, sondern auch die kontinuierliche Kuratierung, Validierung und Kontextualisierung, sodass das LLM bei Bedarf gezielt auf relevantes Wissen zugreifen kann. Dies ermöglicht eine effiziente und skalierbare Erweiterung und Pflege der Wissensbasis analog zu den Lern- und Erinnerungsprozessen eines menschlichen Entwicklers. Im vorliegenden Ansatz wird bewusst darauf verzichtet, das gesamte relevante Wissen direkt im Sprachmodell zu trainieren. Stattdessen bleibt die Wissensbasis extern verfügbar und wird bei Bedarf über Retrieval-Mechanismen integriert. Dies hat mehrere entscheidende Vorteile: Zum einen bleibt das System flexibel gegenüber Modelländerungen, da neue oder leistungsfähigere Sprachmodelle direkt auf die bestehende Wissensbasis zugreifen können, ohne dass ein erneutes aufwändiges Fine-Tuning erforderlich ist. Zum anderen ermöglicht die externe Verwaltung des Wissens eine einfache und gezielte Aktualisierung, Ergänzung oder Korrektur von Informationen, ohne dass das LLM selbst neu trainiert werden muss. Dies erhöht die Wartbarkeit, Transparenz und Anpassbarkeit der Lösung erheblich, insbesondere im industriellen Umfeld, wo sich Maschinenparameter, Unternehmensrichtlinien oder Prozessvorgaben regelmäßig ändern können.

Evaluierung von Steuerungscode (RQ3): Die Verifikation von SPS-Prorammen erfolgt in mehreren Schritten. Zunächst kann die syntaktische Korrektheit des Codes automatisiert durch Compilerprüfungen oder den Einsatz von Lintern sichergestellt werden. Diese Tests erkennen formale Fehler, z. B. eine inkorrekte Syntax nach IEC 61131-3 [3], tragen aber nicht zur Sicherstellung der funktionalen Korrektheit bei. Um die Semantik, d.h. die tatsächliche Funktionsweise des Codes zu überprüfen, werden Unit-Tests (z.B. TcUnit [40]) und Systemtests eingesetzt. Unit-Tests verifizieren einzelne Bausteine oder Programmfunktionen isoliert, während

Systemtests das Zusammenspiel mehrerer Komponenten und deren Reaktion auf verschiedene Eingangssignale untersuchen. Beide Testarten sind unverzichtbar, decken aber physikalische Effekte und komplexes Systemverhalten nur unzureichend ab. Für die umfassende Validierung von Steuerungscode in CPSs bietet daher die VIBN eine erweiterte Möglichkeit. Mit Hilfe von Simulationsplattformen (z.B. ISG Virtuos [41]) kann die Steuerungssoftware in Verbindung mit einem digitalen Modell der realen Anlage unter Echtzeitbedingungen getestet werden. Dabei wird die physikalische Dynamik - wie Bewegungen, Wechselwirkungen oder sicherheitskritische Zustände - realitätsnah simuliert und das Verhalten des Codes im Zielsystem überprüft. Damit ermöglicht VIBN eine frühzeitige und systematische Validierung sowohl der logischen als auch der physikalischen Aspekte des Steuerungscodes, noch bevor die reale Maschine zur Verfügung steht.

Generierung von Feedback aus VIBN Simulation (RQ4): Eine zentrale Herausforderung bei der Integration eines autonomen Evaluierungs- und Verbesserungsprozesses für LLM-generierten Steuerungscode besteht in der Aufbereitung des Feedbacks aus der virtuellen Inbetriebnahme. Testszenarien in der Simulation, z.B. zur Überprüfung der Bewegung eines Werkstücks auf einem Förderband, liefern in der Regel numerische oder boolesche Daten, die den Soll-Ist-Vergleich beschreiben, z.B. Positionsabweichungen innerhalb definierter Toleranzgrenzen. Diese Rohdaten sind jedoch für ein Sprachmodell nicht direkt interpretierbar. Es fehlt eine semantische Übersetzung der Testergebnisse in eine für LLMs verständliche Textform. Um diese Lücke zu schließen, können regelbasierte Auswertungssysteme eingesetzt werden, die auf Basis von Metriken und Zustandsverläufen automatisch strukturierte, beschreibende Rückmeldungen generieren. Solche Systeme interpretieren die Simulationsergebnisse, erkennen Fehlerklassen wie "Werkstück wird zu langsam transportiert" oder "Endposition außerhalb der Toleranz" und wandeln diese in natürlichsprachliche oder maschinenlesbare Beschreibungen um. Dadurch wird es möglich, das LLM gezielt auf Defizite oder Optimierungsbedarf hinzuweisen und eine selbstständige Korrektur- und Lernschleife zu etablieren. Alternativ zur regelbasierten Aufbereitung des Feedbacks besteht die Möglichkeit, Sprachmodelle gezielt so zu trainieren (Fine-Tuning), dass sie Testergebnisse direkt interpretieren können. Durch Training auf geeigneten Datensätzen, die numerische Simulationsausgaben mit entsprechenden semantischen Bewertungen verknüpfen, kann das LLM lernen, Muster und Abweichungen selbstständig zu erkennen und deren Bedeutung korrekt abzuleiten. Auf diese Weise könnte das Modell beispielsweise das Verhalten eines Förderbandes aus Positionsdaten und Zeitreihen selbstständig bewerten und daraus Schlussfolgerungen wie "Fördergeschwindigkeit inkonsistent" oder "Zielposition erreicht" ableiten, ohne auf eine separate regelbasierte Interpretation angewiesen zu

sein. Dieser Ansatz erfordert allerdings umfangreiche und sorgfältig annotierte Trainingsdaten, um die gewünschte semantische Sensitivität gegenüber numerischen Mustern zuverlässig herzustellen.

5. Diskussion und Ausblick

Die Generierung von SPS-Programmen durch LLMs kann bei der Entwicklung von komplexen Automatisierungslösungen unterstützen. Dabei wurde im vorliegenden Artikel ein besonderer Fokus auf die Softwarequalität gelegt. Wesentliche Aspekte des vorgestellten Konzepts sind die Einbindung einer Wissensbasis, die auch die Softwarequalität berücksichtigt, sowie die Verwendung eines LLM-Agenten, der mit der Umwelt interagieren kann und Feedback verständlich für ein Sprachmodell aufbereitet. Ein Kernelement des Konzepts ist die systematische und automatisierte Evaluierung von generierten Steuerungsprogrammen mittels virtueller Inbetriebnahme, um durch iteratives Vorgehen die Qualität des Codes zu verbessern. Die vorgestellten Forschungsfragen und Herausforderungen leiten zukünftige Forschung.

Literaturverzeichnis

[1] Feichtinger, K., Meixner, K., Rinker, F., Koren, I., Eichelberger, H., Heinemann, T., Holtmann, J., Konersmann, M., Michael, J., Neumann, E.-M., Pfeiffer, J., Rabiser, R., Riebisch, M. u. Schmid, K.: Industry Voices on Software Engineering Challenges in Cyber-Physical Production Systems Engineering. 27th International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE 2022

[2] Brauner, P., Dalibor, M., Jarke, M., Kunze, I., Koren, I., Lakemeyer, G., Liebenberg, M., Michael, J., Pennekamp, J., Quix, C., Rumpe, B., van der Aalst, W., Wehrle, K., Wortmann, A. u. Ziefle, M.: A Computer Science Perspective on Digital Transformation in Production. ACM Transactions on Internet of Things 3 (2022) 2

[3] 2013. *IEC 61131-3: Programming industrial automation systems*

[4] Wiesmayr, B., Zoitl, A. u. Rabiser, R.: Assessing the usefulness of a visual programming IDE for large-scale automation software. Software and Systems Modeling 22 (2023) 5, S. 1619–1643

[5] Sonnleithner, L., Wiesmayr, B., Gutiérrez, A. M., Rabiser, R. u. Zoitl, A.: Complexity of Structured Text in IEC 61499 Function Blocks: A Survey. 28th International Conference on Emerging Technologies and Factory Automation (ETFA). 2023

[6] Vogel-Heuser, B. u. Ocker, F.: Maintainability and evolvability of control software in machine and plant manufacturing — An industrial survey. Control Engineering Practice 80 (2018), S. 157–173

[7] Oberlehner, M., Sonnleithner, L., Wiesmayr, B. u. Zoitl, A.: Catalog of Refactoring Operations for IEC 61499. 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). 2021

[8] Fadhlillah, H. S., Wiesmayr, B., Oberlehner, M., Rabiser, R. u. Zoitl, A.: Towards Delta-Oriented Variability Modeling for IEC 61499. 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). 2021

[9] Hölldobler, K., Michael, J., Ringert, J. O., Rumpe, B. u. Wortmann, A.: Innovations in Model-based Software and Systems Engineering. The Journal of Object Technology 18 (2019) 1

[10] Wortmann, A., Combemale, B. u. Barais, O.: A Systematic Mapping Study on Modeling for Industry 4.0. ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS). ACM 2017, S. 281–291

[11] Koziolek, H., Burger, A., Platenius-Mohr, M. u. Jetley, R.: A classification framework for automated control code generation in industrial automation. Journal of Systems and Software 166 (2020)

[12] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. u. Polosukhin, I.: Attention is All you Need. Advances in Neural Information Processing Systems 30 (2017)

[13] Tran, K., Zhang, J., Pfeiffer, J., Wortmann, A. u. Wiesmayr, B.: Generating PLC Code with Universal Large Language Models. 29th International Conference on Emerging Technologies and Factory Automation (ETFA). 2024

[14] Koziolek, H., Gruener, S. u. Ashiwal, V.: ChatGPT for PLC/DCS Control Logic Generation. 28th International Conference on Emerging Technologies and Factory Automation (ETFA). 2023

[15] OpenAI: GPT-4. openai.com/index/gpt-4/, abgerufen am: 02.05.2025

[16] anthropic: Claude 3.5 Sonnet. anthropic.com/news/claude-3-5-sonnet, abgerufen am: 02.05.2025

[17] Meta: Llama. llama.com/models/llama-4, abgerufen am: 02.05.2025

[18] deepseek: DeepSeek V3. deepseek.com, abgerufen am: 02.05.2025

[19] Min, B., Ross, H., Sulem, E., Veyseh, A. P. B., Nguyen, T. H., Sainz, O., Agirre, E., Heintz, I. u. Roth, D.: Recent Advances in Natural Language Processing via Large Pre-trained Language Models: A Survey. ACM Computing Surveys 56 (2024) 2

[20] Zhang, B., Li, C. u. Fan, K.: MARIO Eval: Evaluate Your Math LLM with your Math LLM--A mathematical dataset evaluation toolkit, arXiv 2024. arxiv.org/pdf/2404.13925v1

[21] Li, G., Hammoud, H., Itani, H., Khizbullin, D. u. Ghanem, B.: CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society. Advances in Neural Information Processing Systems 36 (2023), S. 51991–52008

[22] Marvin, G., Hellen, N., Jjingo, D. u. Nakatumba-Nabende, J.: Prompt Engineering in Large Language Models. Data Intelligence and Cognitive Informatics. Proceedings of ICDICI 2023. Algorithms for Intelligent Systems. Singapore: Springer 2024, S. 387–402

[23] Wang, Y., Yao, Q., Kwok, J. T. u. Ni, L. M.: Generalizing from a Few Examples: A Survey on Few-shot Learning. ACM Computing Surveys 53 (2020) 3

[24] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q. V. u. Zhou, D.: Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Advances in Neural Information Processing Systems 35 (2022), S. 24824–24837

[25] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S. u. Kiela, D.: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Advances in Neural Information Processing Systems 33 (2020), S. 9459–9474

[26] Han, Z., Gao, C., Liu, J., Zhang, J. u. Zhang, S. Q.: Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey, arXiv 2024. arxiv.org/pdf/2403.14608

[27] Hu, J., Liao, X., Gao, J., Qi, Z., Zheng, H. u. Wang, C.: Optimizing Large Language Models with an Enhanced LoRA Fine-Tuning Algorithm for Efficiency and Robustness in NLP Tasks, arXiv 2024. arxiv.org/pdf/2412.18729

[28] Agarwal, B. B., Tayal, S. P. u. Gupta, M.: Software engineering and testing. Computer Science Series. Jones and Bartlett Publishers 2009

[29] Wünsch, G.: Methoden für die virtuelle Inbetriebnahme automatisierter Produktionssysteme, Bd. 215. München: Herbert Utz Verlag 2008

[30] Lacour, F.-F.: Modellbildung für die physikbasierte Virtuelle Inbetriebnahme materialflussintensiver Produktionsanlagen, Bd. 257. München: Herbert Utz Verlag 2012

[31] Koziolek, H., Grüner, S., Hark, R., Ashiwal, V., Linsbauer, S. u. Eskandani, N.: LLM-based and Retrieval-Augmented Control Code Generation. 2024 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code). 2024, S. 22–29

[32] Beckhoff Automation GmbH u. Co. KG: TwinCAT 3, 2025. www.beckhoff.com/de-de/produkte/automation/twincat/, abgerufen am: 02.05.2025

[33] Chroma: Chroma, 2025. trychroma.com, abgerufen am: 02.05.2025

[34] PostgreSQL: PostgreSQL, 2025. https://www.postgresql.org/, abgerufen am: 28.04.2025

[35] Neo4j, Inc.: Neo4j Graph Database & Analytics, 2024. neo4j.com, abgerufen am: 02.05.2025

[36] OWL Working Group: OWL - Semantic Web Standards, 2012. www.w3.org/OWL, abgerufen am: 02.05.2025
[37] Alt, O.: Modellbasierte Systementwicklung mit SysML. München: Hanser 2012
[38] OPC Foundation: The industrial interoperability standard, 2023. opcfoundation.org, abgerufen am: 02.05.2025
[39] IDTA: Der Standard für den Digitalen Zwilling, 2025. industrialdigitaltwin.org, abgerufen am: 02.05.2025
[40] TcUnit, 2024. tcunit.org, abgerufen am: 02.05.2025
[41] ISG Industrielle Steuerungstechnik GmbH: ISG-virtuos, 2022. www.isg-stuttgart.de/produkte/softwareprodukte/isg-virtuos, abgerufen am: 02.05.2025