An Agentic Approach to Automatic Creation of P&ID Diagrams from Natural Language Descriptions

Shreeyash Gowaikar[1], **Srinivasan Iyengar**[2], **Sameer Segal**[1], **Shivkumar Kalyanaraman**[2]

[1]Microsoft Research India
[2]Mircosoft Corporation

Abstract

The Piping and Instrumentation Diagrams (P&IDs) are foundational to the design, construction, and operation of workflows in the engineering and process industries. However, their manual creation is often labor-intensive, error-prone, and lacks robust mechanisms for error detection and correction. While recent advancements in Generative AI, particularly Large Language Models (LLMs) and Vision-Language Models (VLMs), have demonstrated significant potential across various domains, their application in automating generation of engineering workflows remains underexplored. In this work, we introduce a novel copilot for automating the generation of P&IDs from natural language descriptions. Leveraging a multi-step agentic workflow, our copilot provides a structured and iterative approach to diagram creation directly from Natural Language prompts. We demonstrate the feasibility of the generation process by evaluating the soundness and completeness of the workflow, and show improved results compared to vanilla zero-shot and few-shot generation approaches.

Introduction

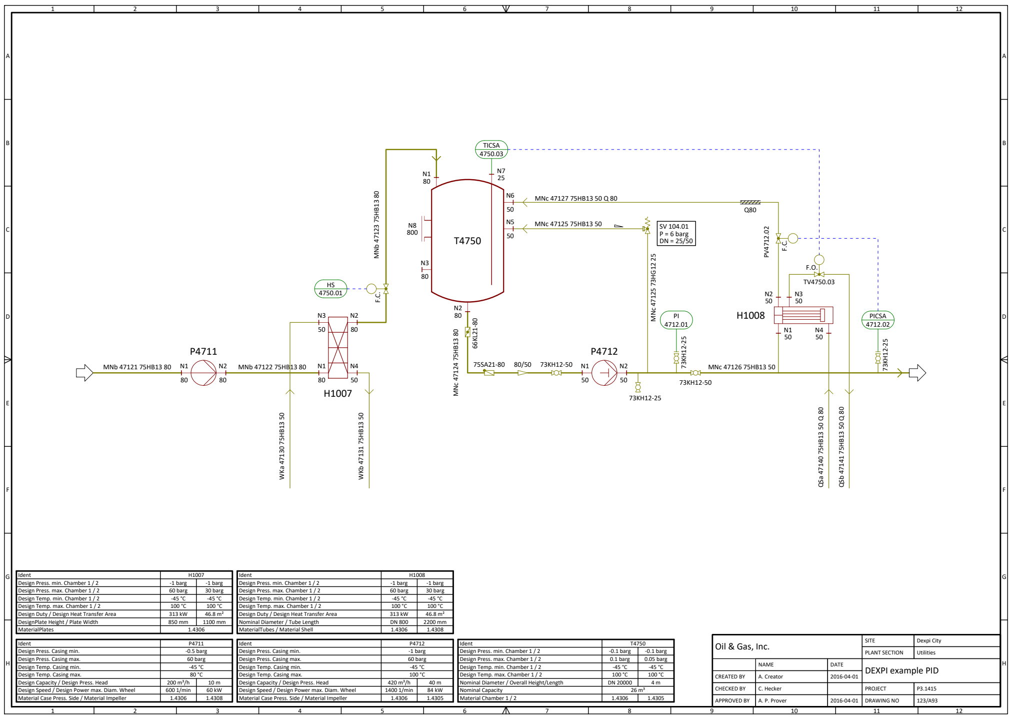
Engineering Diagrams (EDs) such as Block Flow Diagrams (BFDs), Process Flow Diagrams (PFDs) and Piping and Instrumentation Diagrams (P&IDs) are essential for the design, construction, operation, and maintenance of chemical and process plants (Sakhinana, Sannidhi, and Runkana 2024). They play a crucial role in the effective dissemination of information to all stakeholders in the chemical, energy and process industry. BFDs show the preliminary design of unit operations, process flows, inputs, and outputs, while PFDs present major equipment, flow paths, key instruments, and some process details like flow rates and contents, and P&IDs offer a more detailed view, showing all equipment, pipes, and instruments in a plant (Nasby 2012). However, the process of creating PFDs and P&IDs manually, relying on schemes from previous projects, design guidelines, engineer expertise, and Computer-Aided Design (CAD) software, can often become tedious and error-prone (Hirtreiter, Schulze Balhorn, and Schweidtmann 2023). In addition to being laborious, due to the manual creation, the entire diagram generation process has minimal provenance for error detection and correction.

In the past few years, Generative AI has achieved a remarkable progress. Large Language Models (LLMs) now excel in tasks like Text Classification, Question Answering, Text Summarising, Text Translation, Machine Translation, Code Generation, Text Generation, and Sentiment Analysis (Laskar et al. 2023). Multi modal Vision Language Models (VLMs), have also progressed to solve tasks like Visual Reasoning, Visual Question Answering and Image Generation (Zhang et al. 2024). Attributed to this progress, Generative AI has been adopted for various applications and automations across a myriad of industries and domains (Gozalo-Brizuela and Garrido-Merchán 2023).

Artificial Intelligence (AI) has been employed in the chemical and process industry for over 35 years and it has achieved notable successes (Venkatasubramanian 2019). The earliest use of AI in process systems engineering, include the AIDES (Adaptive Initial DEsign Synthesizer system) (Siirola and Rudd 1971), that uses end-to-end analysis, linear programming and symbol matching for chemical process generation, the Design-Kit (Stephanopoulos et al. 1987), a rule-based logical program for flow sheet synthesis and operational analysis, MODELL.LA (Stephanopoulos, Henning, and Leone 1990), a modelling language for process system models, and MODEX (Rich and Venkatasubramanian 1987), a causal model based system for fault diagnosis.

More recently, Machine Learning (ML), Deep Learning (DL) and Reinforcement Learning (RL) algorithms are being used to solve classical problems in the process industry. There are efforts to use ML, DL, and RL algorithms to digitize classical P&IDs in paper or PDF format (Paliwal et al. 2021; Kim et al. 2022), predict next element in PFDs and P&IDs (Vogel, Schulze Balhorn, and Schweidtmann 2023), translate PFDs to P&IDs (Hirtreiter, Schulze Balhorn, and Schweidtmann 2024), generate control code (Koziolek and Koziolek 2023), and even do question answering on P&IDs (Sakhinana, Sannidhi, and Runkana 2024). However, we find a gap in existing literature to automate generation of P&IDs directly from Natural Language. Our work serves as a preliminary to the field of generating P&IDs directly from Natural Language utterances.

Through this work, we introduce a novel application of Generative AI for the automatic, sub-system level generation of P&IDs. We propose a copilot powered by a multi-step



Ident	H1007	
Design Press. min. Chamber 1 / 2	-1 barg	-1 barg
Design Press. max. Chamber 1 / 2	60 barg	30 barg
Design Temp. min. Chamber 1 / 2	-45 °C	-45 °C
Design Temp. max. Chamber 1 / 2	100 °C	100 °C
Design Duty / Design Heat Transfer Area	313 kW	46.8 m²
DesignPlate Height / Plate Width	850 mm	1100 mm
MaterialPlates	1.4306	

Ident	H1008	
Design Press. min. Chamber 1 / 2	-1 barg	-1 barg
Design Press. max. Chamber 1 / 2	60 barg	30 barg
Design Temp. min. Chamber 1 / 2	-45 °C	-45 °C
Design Temp. max. Chamber 1 / 2	100 °C	100 °C
Design Duty / Design Heat Transfer Area	313 kW	46.8 m²
Nominal Diameter / Tube Length	DN 800	2200 mm
MaterialTubes / Material Shell	1.4306	1.4308

Ident	P4711	
Design Press. Casing min.	-0.5 barg	
Design Press. Casing max.	60 barg	
Design Temp. Casing min.	-45 °C	
Design Temp. Casing max.	80 °C	
Design Capacity / Design Press. Head	200 m³/h	10 m
Design Speed / Design Power max. Diam. Wheel	600 1/min	60 kW
Material Case Press. Side / Material Impeller	1.4306	1.4308

Ident	P4712	
Design Press. Casing min.	-1 barg	
Design Press. Casing max.	60 barg	
Design Temp. Casing min.	-45 °C	
Design Temp. Casing max.	100 °C	
Design Capacity / Design Press. Head	420 m³/h	40 m
Design Speed / Design Power max. Diam. Wheel	1400 1/min	84 kW
Material Case Press. Side / Material Impeller	1.4306	1.4305

Ident	T4750	
Design Press. min. Chamber 1 / 2	-0.1 barg	-0.1 barg
Design Press. max. Chamber 1 / 2	0.1 barg	0.05 barg
Design Temp. min. Chamber 1 / 2	-45 °C	-45 °C
Design Temp. max. Chamber 1 / 2	100 °C	100 °C
Nominal Diameter / Overall Height/Length	DN 20000	4 m
Nominal Capacity	26 m³	
Material Chamber 1 / 2	1.4306	1.4305

Oil & Gas, Inc.			SITE	Dexpi City
			PLANT SECTION	Utilities
	NAME	DATE	DEXPI example PID	
CREATED BY	A. Creator	2016-04-01		
CHECKED BY	C. Hecker		PROJECT	P3.1415
APPROVED BY	A. P. Prover	2016-04-01	DRAWING NO	123/A93

Figure 1: An example of a simple P&ID sheet (DEXPI Consortium 2020b). It entails Equipments (Tank, Heat Exchangers and pumps), Piping Systems, Instruments (Globe Valves, Ball Valves, Pipe Reducers, Butterfly Valve, Spring Loaded Globe Safety Valve and Swing Check Valves), Pipe Off Page Connectors and Actuating Systems.

agentic workflow to generate a P&ID from only its linguistic description. We nickname the copilot - the ACPID Copilot as an acronym for the **A**utomatic **C**reation of **P&ID**s. Aimed at improving efficiency and productivity of the engineers, the ACPID copilot also improves provenance of the generation process by enabling audit trails. The copilot outputs a textual representation of the P&ID, a Microsoft Visio Diagram of the P&ID for engineers to edit during the generation process, and also a natural language description of the current P&ID that can be append in subsequent prompts for iterative development through multi-turn conversations. The copilot's design also allows users to edit and start with an existing P&ID, facilitating quick adoption and integration into existing workflows. We have organised the paper into the following sections for easier understanding:

- The *Background* section provides the essential background information.
- The *Methodolgy* section describes in detail the entire process design.
- The *Evaluations* section describes the evaluation strategy and provides the results of our approach.
- We conclude with the *Discussions* and *Conclusions* section, which talks about the impact, limitations and future avenues of our research.

Background

P&ID

Piping and Instrumentation Diagrams (P&IDs) are critical tools for the design, operation, and maintenance of complex process systems, including those found in chemical plants, refineries, and power stations. An example of a P&ID diagram can be seen in Figure 1. These diagrams serve as blueprints, providing a comprehensive view of the processes, equipment, and instrumentation They are integral to various aspects of system management, including financial planning, safety considerations, and operational efficiency.

Financial Impact P&IDs directly influence financial decisions by enabling the creation of accurate Bills of Material (BoM) and optimizing supply chain planning.

Safety Considerations P&IDs play a crucial role in ensuring safety, particularly in environments where hazardous materials or extreme conditions are involved. By detailing the flow of materials and equipment interconnections, P&IDs help in developing safety protocols and hazard mitigation strategies.

Operational Planning P&IDs are essential for managing ongoing operations and maintenance by ensuring all system components are well-documented for efficient decision-making. They also aid in planning project timelines and inventories, and are particularly valuable for retrofitting or upgrading facilities, providing a clear reference for modifications with minimal disruption.

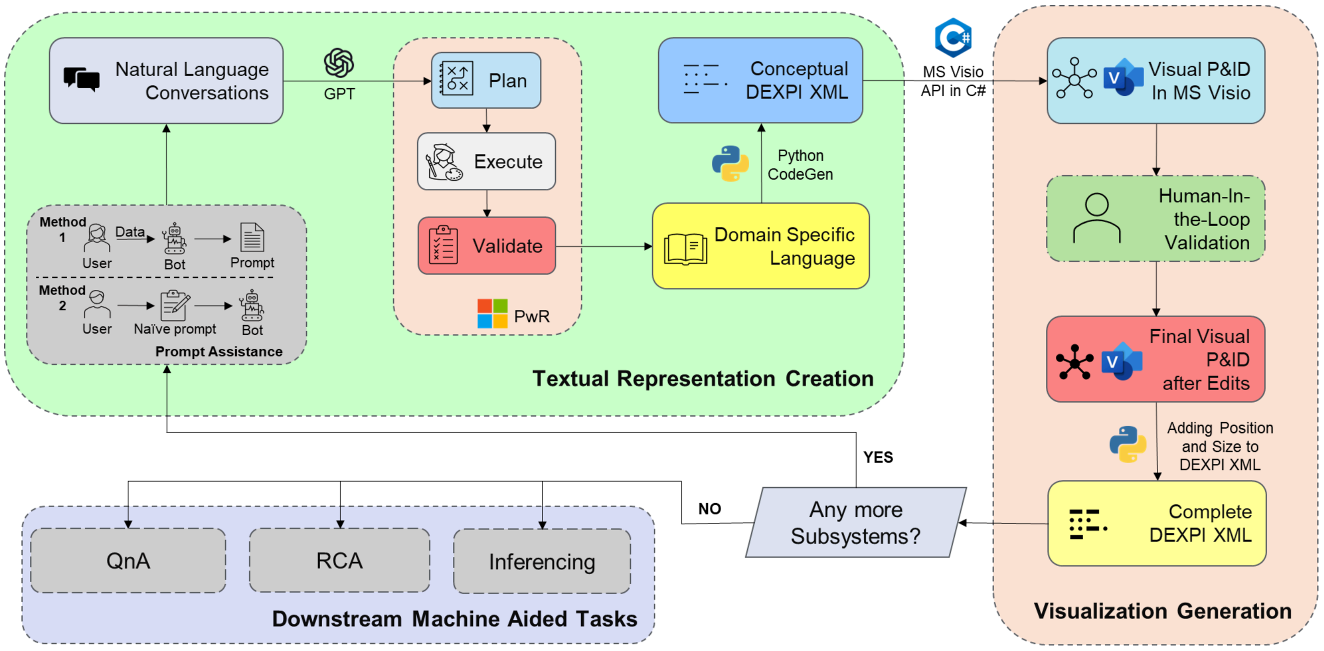


Figure 2: The architectural block diagram of the ACPID Copilot

P&IDs include diagrammatic information on equipments (such as reactors, centrifuges, heat exchangers, pumps, etc), piping systems that transport materials and information, control instruments for measuring and regulating process variables, and actuating systems that adjust flows and pressures based on control signals. Additionally, each P&ID is also associated with one or many data sheets that detail out the material which is stored and processed in each pipe or equipment, along with its physical and chemical attributes. The data sheets are generally standalone documents but can also be included in the diagram similar to Figure 1 (see left bottom corner).

Data EXchange in the Process Industry (DEXPI)

Engineers use different Computer Aided Engineering (CAE) software to view, edit and create P&IDs. However, each software tool has its own data handling framework and there exists a lack of interoperability between CAE (and other) systems (Wiedau et al. 2019). In executing such projects that encompass the planning, construction, and operation of process plants, companies encounter significant challenges in data exchange. A primary contributor to these challenges is the lack of a universally adopted data exchange standard within the process industry, leading to incompatible systems and increased effort for data management (DEXPI Consortium 2020a). To enhance efficiency and digital interoperability, the *Data EXchange in the Process Industry (DEXPI)* initiative was established. The DEXPI initiative integrates concepts from various existing standards such as the International Organization for Standardization (ISO), International Electro technical Commission (IEC), BIM, Open Platform Communications Unified Architecture (OPC), Capital Facilities Information Handover Specification (CFIHOS) or NAMUR organization recommendations (Wiedau et al. 2019), and implements them based upon the ISO 15962 specification (International Organization for Standardization 2022). It offers a textual, machine-readable, and extendable representation of P&IDs, facilitating interoperability.

The exchange format of the most recent DEXPI standard (DEXPI standard 1.3) is the Proteus Schema 4.1 (ProteusXML 2024), that ensures a mapping from DEXPI elements to XML patterns. It provides an efficient way to encode P&ID diagrams into XML files following the Proteus Schema. A general DEXPI-compliant Proteus XML, referred to hereafter as the DEXPI XML, has the DEXPI Model as the root of the XML composition hierarchy. In the Proteus XML Schema the DEXPI Model is depicted as the *Plant Model* XML Class and can be seen in Figure 3. The DEXPI Model contains three important parts for a coherent and complete DEXPI XML representation of a P&ID:

- *Conceptual Model*: This contains all the engineering information of the DEXPI model like elements, connections and their attributes, and is independent of the drawing.
- *Drawing*: This class contains all the drawing information like the border, page dimensions or page color.
- *Shape Catalogue*: This contains shape information of all the elements. It describes how shapes and symbols are created using lines and arcs.

Architecture

The aim of the ACPID copilot is to create P&IDs from Natural Language prompts. However, due to intricate design of P&IDs, framing it as a Visual Generation task becomes difficult. Hence, we frame the problem as a Natural Language

Generation task to create the DEXPI XML textual representation of the P&ID from the prompts. We then transform the textual representation into CAE based visual diagrams, rendered with Microsoft Visio, using deterministic rules.

We propose a novel workflow incorporating Plan and Execute Agents (Wang et al. 2023) and rule-based synthesis for the generation of P&IDs. The agent-based system organizes and structures information in a format optimized for rule-based synthesis. The output of the agent-based aggregator is then translated into the DEPXI Proteus XML representation through a deterministic, rule-based translation process. The architecture of the ACPID copilot (Figure 2) can be divided into three main modules—the Textual Representation Creation, Visualization Generation, and Downstream tasks. The following subsections dive deeper into each of these three parts.

Textual Representation Creation

The *Textual Representation Creation* module translates natural language conversations describing P&ID components into a DEXPI XML representation. This module's architecture is adapted from Microsoft's open-sourced Programming with Representation (PwR) framework (YM et al. 2023), in which natural language prompts are converted to a Domain-Specific Language (DSL) and subsequently translated deterministically into code, enabling robust, interpretable, and efficient code generation. The intermediate DSL serves as a JSON equivalent of a Finite State Machine, outlining the workflow needed to integrate the element into the system. We modify the DSL generation and deterministic code generation components to meet the requirements for DEXPI generation. The entire architecture is powered by a multi-step agentic workflow over a pre-trained LLM. The multi-step agentic workflow can be broken into the following main sub-modules:

Planning The first step of the agentic workflow is creation of an execution plan from the user provided prompt. The planning step involves creation of steps that are to be performed by an LLM to add an element (equipment, instrument or actuating system) or a connection.

Execution Once the plan is generated, we iterate over the planned steps and execute each step using an LLM. Each execution-step prompt contains the description of the step generated during the planning phase, and the utterance from the original user prompt that corresponds to the planned step. For each step, to provide context for existing elements and connections, we append outputs of all previously executed plan-steps with the prompt. The LLM which executes the plan can be different than the LLM which generates the plan.

Validation and Pruning When all the planned steps are executed, we deploy rule-based validation to validate the flow of the plan and execution steps, and prune unnecessary floating steps. Currently, the validation only verifies the flow and transitions, but can be modified in the future to add P&ID validation support for the generated content. The output of this step, consists of information derived from the prompt which is then expanded, formatted, and organized in a custom domain-specific language (DSL), which is essential for further processing.

Domain Specific Language (DSL) The DSL output from the preceding sub-module acts as an intermediate representation before we convert it into the DEXPI XML. We propose that the conversion through an intermediate representation, like the DSL, that imposes fewer syntactic constraints compared to the final DEXPI XML, improves the quality of the translation. The DSL is specifically designed to encapsulate structured information derived from the input prompt, facilitating its organized and efficient rule-based conversion into DEXPI XML.

Translate to DEXPI To generate the DEXPI XML representation of the P&ID, we employ a deterministic, rule-based translation of the DSL. We employ predefined mapping, enabled by the encapsulation of information in the DSL, to improve consistency and reliability. Moreover, this determinism also helps improve the transparency and adaptability of the generation process.

Visual Diagram Generator

The preceding *Textual Representation Creation Module* generates only the DEXPI Conceptual Model of the P&ID, without including any visual elements. This textual output is subsequently utilized to construct the visual representation of the P&ID. The *Visual Diagram Generator* module then incorporates this visual information, enriching the DEXPI XML with data for the Shape Catalogue and the Diagram class.

The Conceptual Model generated by the Textual Representation Creation module is parsed and converted into a Microsoft Visio Drawing via the Microsoft Visio C# API, serving as an initial draft for human validation and refinement. The **Human-in-the-Loop** (HITL) postulation in the workflow provides engineers with the ability to verify the accuracy of elements and connections within the generated P&ID. Additionally, HITL enables engineers to manually adjust the diagram according to project-specific preferences, personal preferences, and prior design standards. Ultimately, HITL promotes robustness and responsible AI practices throughout the generation phase.

Once the engineer is satisfied with the Drawing, we extract the position and shape information and insert it into the existing DEXPI XML output from the first module. We complete the DEXPI XML by adding position information to necessary elements and the shape information to the Drawing and Shape Catalogue classes in the DEXPI XML.

Downstream Tasks and the Iterative Workflow

The ACPID copilot is designed for subsystem-level generation, allowing engineers to iteratively loop through the entire workflow for adding more subsystems while efficiently reusing already generated subsystems. This approach emulates the current manual process, where engineers build the system step-by-step, focusing on one subsystem at a time. Subsystem-level generation also enables engineers to validate and refine their prompts more effectively, while optimizing the ACPID copilot's performance by maintaining manageable prompt lengths (Levy, Jacoby, and Goldberg 2024). This capability for subsystem-level generation is what enables users to modify existing P&IDs.

If no additional subsystems remain, our workflow enables engineers to perform various downstream tasks, including Question Answering, Root Cause Analysis (RCA), and summarization. This is due to the outputs and digital by-products generated throughout the workflow. The output of the Visualization Generation Module includes a Visual P&ID, in the Visio Drawing format (*.vsdx*), and a complete DEXPI XML including the Conceptual Model, Shape Catalogue and Drawing information. During the Textual Representation Creation, we also obtain a Natural Language Description of the entire P&ID, by parsing the DEXPI XML. This description is generated for enabling multi-turn conversations during generation, and hence can be used optimally for downstream tasks as well. As a result, due to the interoperability offered by DEXPI standards, and a spectrum of digital products created as outputs or by-products of the workflow, we enable multiple downstream tasks for the engineers.

Evaluations

Dataset: To evaluate the performance of The ACPID copilot, we create an evaluation test-bench from the DEXPI examples (DEXPI Consortium 2022) provided by the DEXPI consortium. We create tests to evaluate the soundness of the generation and the completeness of the DEXPI Proteus XML creation.

Performance Metrics: Aligned with its mathematical definition, we define the *soundness* of the generation process as the property that ensures every element referenced in the prompt is included in the response in a structured and coherent manner. Similarly, we define *completeness* of the generation process, as the existence of all fields required for a DEXPI XML to be considered syntactically complete and to support interoperability.

Baseline: We compare the results with zero-shot and few-shot performances of GPT-4-Turbo model which also acts as the base model for our copilot. We create few-shot prompts by appending actual DEXPI XML outputs provided from the DEXPI Examples for elements and connections. Care is taken to avoid using same examples during few-shot generation. Figure 3 presents examples of outputs generated by The ACPID copilot, alongside those produced by the zero-shot and few-shot approaches for comparison.

Soundness

Mathematically, if $Elements$ is the set of all equipments, instruments and actuating systems, $Connections$ is a set describing connections between two elements and $Attributes$ is the set of all attributes for any connection or element, for an element $e \in Elements$, a connection $c \in Connections$ and an attribute $a \in Attributes$, we evaluate soundness as the proportion of inference-instances in which $e \; or \; c \; or \; a$ appears in both the prompt and the generated DEXPI XML. We report $Soundness$, as percentage responses that contain the element e or the connection c or the attribute a, whenever the corresponding prompt mentions $e \; or \; c \; or \; a$.

Our evaluation is based only on the DEXPI XML, as the visual diagram is a deterministic derivation of the XML in our workflow. We extract about *132* artifacts generated by elements, connections and attributes with a wide coverage over different scenarios, to evaluate soundness while generation. We assess the soundness of the approach, irrespective of the completeness and correctness of the generated DEXPI XML. Therefore, we also include incorrectly generated DEXPI XML samples and focus solely on evaluating whether the response incorporates the elements from the prompt in any structured manner. We chose this evaluation approach based on the observation that both zero-shot and few-shot approaches fail to produce credible DEXPI XML.

Method	**Soundness**
Zero-Shot	58.33%
Few-Shot	65.90%
ACPID Copilot	**96.96%**

Table 1: Comparing soundness during generation, between Zero-shot, Few-shot generation from GPT-4-Turbo and the ACPID copilot.

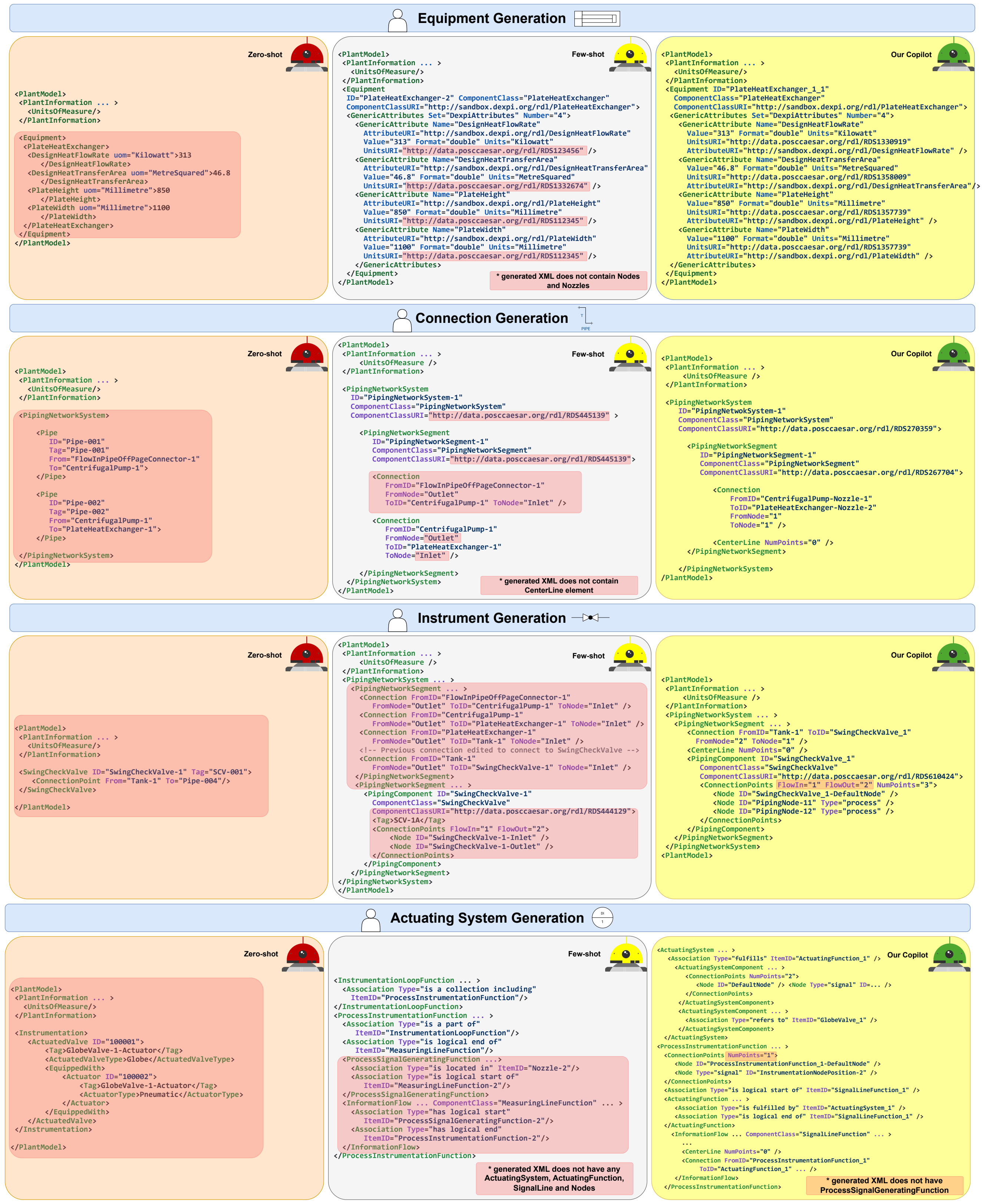
The results can be seen in the Table 1, where the ACPID copilot clearly outperforms the few-shot and zero-shot variations.

Completeness

We identify sections of the DEXPI XML necessary to visualize and enable interoperability of the DEXPI XML across platforms. For equipments and instruments, we check the correctness of the XML Class (*Element*), and its Attributes. We then check the existence of the ID, the Component Name, the Component Class (and correctness of *ComponentClassURL*), the Nozzles and Nodes, and the Position and Scale sub-elements. For Connections, we check the correctness of the XML Class (*PipingNetworkSystem, PipingNetworkSegment*), and the Connection element that specifies the source and destination element. We then check the existence of CenterLine (element required for depiction of lines in the visual DEXPI XML). Finally, for the actuating systems, we again evaluate the correctness of the XML Class (*ActuatingSystem*) and check the existence of associated *InstrumentationLoopFunction, ProcessInstrumentationFunction* with *Actuating Function*, and *InformationFlows* while verifying the correctness of necessary *Associations* elements. We test the completeness by evaluating on *555* sections extracted from a DEXPI Example provided by the DEXPI Consortium.

The results can be seen in the Table 2, where our copilot outperforms the zero-shot and few-shot variations by a large margin.

We observe that the vanilla zero-shot generation is entirely unable to create a complete and syntactically valid DEXPI XML even though GPT-4-Turbo has knowledge



Equipment Generation — Zero-shot

```xml
<PlantModel>
 <PlantInformation ... >
   <UnitsOfMeasure/>
 </PlantInformation>

 <Equipment>
  <PlateHeatExchanger>
   <DesignHeatFlowRate uom="Kilowatt">313
      </DesignHeatFlowRate>
   <DesignHeatTransferArea uom="MetreSquared">46.8
      </DesignHeatTransferArea>
   <PlateHeight uom="Millimetre">850
      </PlateHeight>
   <PlateWidth uom="Millimetre">1100
      </PlateWidth>
  </PlateHeatExchanger>
 </Equipment>
</PlantModel>
```

Equipment Generation — Few-shot

```xml
<PlantModel>
 <PlantInformation ... >
   <UnitsOfMeasure/>
 </PlantInformation>
 <Equipment
  ID="PlateHeatExchanger-2" ComponentClass="PlateHeatExchanger"
  ComponentClassURI="http://sandbox.dexpi.org/rdl/PlateHeatExchanger">
  <GenericAttributes Set="DexpiAttributes" Number="4">
    <GenericAttribute Name="DesignHeatFlowRate"
      AttributeURI="http://sandbox.dexpi.org/rdl/DesignHeatFlowRate"
      Value="313" Format="double" Units="Kilowatt"
      UnitsURI="http://data.posccaesar.org/rdl/RDS123456" />
    <GenericAttribute Name="DesignHeatTransferArea"
      AttributeURI="http://sandbox.dexpi.org/rdl/DesignHeatTransferArea"
      Value="46.8" Format="double" Units="MetreSquared"
      UnitsURI="http://data.posccaesar.org/rdl/RDS1332674" />
    <GenericAttribute Name="PlateHeight"
      AttributeURI="http://sandbox.dexpi.org/rdl/PlateHeight"
      Value="850" Format="double" Units="Millimetre"
      UnitsURI="http://data.posccaesar.org/rdl/RDS112345" />
    <GenericAttribute Name="PlateWidth"
      AttributeURI="http://sandbox.dexpi.org/rdl/PlateWidth"
      Value="1100" Format="double" Units="Millimetre"
      UnitsURI="http://data.posccaesar.org/rdl/RDS112345" />
    </GenericAttributes>
  </Equipment>
</PlantModel>
```

* generated XML does not contain Nodes and Nozzles

Equipment Generation — Our Copilot

```xml
<PlantModel>
 <PlantInformation ... >
   <UnitsOfMeasure/>
 </PlantInformation>
 <Equipment ID="PlateHeatExchanger_1_1"
   ComponentClass="PlateHeatExchanger"
   ComponentClassURI="http://sandbox.dexpi.org/rdl/PlateHeatExchanger">
  <GenericAttributes Set="DexpiAttributes" Number="4">
    <GenericAttribute Name="DesignHeatFlowRate"
      Value="313" Format="double" Units="Kilowatt"
      UnitsURI="http://data.posccaesar.org/rdl/RDS1330919"
      AttributeURI="http://sandbox.dexpi.org/rdl/DesignHeatFlowRate" />
    <GenericAttribute Name="DesignHeatTransferArea"
      Value="46.8" Format="double" Units="MetreSquared"
      UnitsURI="http://data.posccaesar.org/rdl/RDS1358009"
      AttributeURI="http://sandbox.dexpi.org/rdl/DesignHeatTransferArea"/>
    <GenericAttribute Name="PlateHeight"
      Value="850" Format="double" Units="Millimetre"
      UnitsURI="http://data.posccaesar.org/rdl/RDS1357739"
      AttributeURI="http://sandbox.dexpi.org/rdl/PlateHeight" />
    <GenericAttribute Name="PlateWidth"
      Value="1100" Format="double" Units="Millimetre"
      UnitsURI="http://data.posccaesar.org/rdl/RDS1357739"
      AttributeURI="http://sandbox.dexpi.org/rdl/PlateWidth" />
  </GenericAttributes>
 </Equipment>
</PlantModel>
```

Connection Generation — Zero-shot

```xml
<PlantModel>
 <PlantInformation ... >
   <UnitsOfMeasure/>
 </PlantInformation>

 <PipingNetworkSystem>

     <Pipe
        ID="Pipe-001"
        Tag="Pipe-001"
        From="FlowInPipeOffPageConnector-1"
        To="CentrifugalPump-1">
     </Pipe>

     <Pipe
        ID="Pipe-002"
        Tag="Pipe-002"
        From="CentrifugalPump-1"
        To="PlateHeatExchanger-1">
     </Pipe>

 </PipingNetworkSystem>
</PlantModel>
```

Connection Generation — Few-shot

```xml
<PlantModel>
 <PlantInformation ... >
     <UnitsOfMeasure />
 </PlantInformation>

 <PipingNetworkSystem
   ID="PipingNetworkSystem-1"
   ComponentClass="PipingNetworkSystem"
   ComponentClassURI="http://data.posccaesar.org/rdl/RDS445139" >

     <PipingNetworkSegment
        ID="PipingNetworkSegment-1"
        ComponentClass="PipingNetworkSegment"
        ComponentClassURI="http://data.posccaesar.org/rdl/RDS445139">

        <Connection
           FromID="FlowInPipeOffPageConnector-1"
           FromNode="Outlet"
           ToID="CentrifugalPump-1" ToNode="Inlet" />

        <Connection
           FromID="CentrifugalPump-1"
           FromNode="Outlet"
           ToID="PlateHeatExchanger-1"
           ToNode="Inlet" />

     </PipingNetworkSegment>
   </PipingNetworkSystem>
</PlantModel>
```

* generated XML does not contain CenterLine element

Connection Generation — Our Copilot

```xml
<PlantModel>
 <PlantInformation ... >
     <UnitsOfMeasure />
 </PlantInformation>

 <PipingNetworkSystem
    ID="PipingNetwokSystem-1"
    ComponentClass="PipingNetworkSystem"
    ComponentClassURI="http://data.posccaesar.org/rdl/RDS270359">

      <PipingNetworkSegment
         ID="PipingNetworkSegment-1"
         ComponentClass="PipingNetworkSegment"
         ComponentClassURI="http://data.posccaesar.org/rdl/RDS267704">

            <Connection
                FromID="CentrifugalPump-Nozzle-1"
                ToID="PlateHeatExchanger-Nozzle-2"
                FromNode="1"
                ToNode="1" />

            <CenterLine NumPoints="0" />
      </PipingNetworkSegment>

    </PipingNetworkSystem>
/PlantModel>
```

Instrument Generation — Zero-shot

```xml
<PlantModel>
 <PlantInformation ... >
   <UnitsOfMeasure/>
 </PlantInformation>

 <SwingCheckValve ID="SwingCheckValve-1" Tag="SCV-001">
    <ConnectionPoint From="Tank-1" To="Pipe-004"/>
 </SwingCheckValve>

</PlantModel>
```

Instrument Generation — Few-shot

```xml
<PlantModel>
 <PlantInformation ... >
     <UnitsOfMeasure />
 </PlantInformation>
 <PipingNetworkSystem ... >
  <PipingNetworkSegment ... >
   <Connection FromID="FlowInPipeOffPageConnector-1"
      FromNode="Outlet" ToID="CentrifugalPump-1" ToNode="Inlet" />
   <Connection FromID="CentrifugalPump-1"
      FromNode="Outlet" ToID="PlateHeatExchanger-1" ToNode="Inlet" />
   <Connection FromID="PlateHeatExchanger-1"
      FromNode="Outlet" ToID="Tank-1" ToNode="Inlet" />
   <!-- Previous connection edited to connect to SwingCheckValve -->
   <Connection FromID="Tank-1"
      FromNode="Outlet" ToID="SwingCheckValve-1" ToNode="Inlet" />
  </PipingNetworkSegment>
  <PipingNetworkSegment ... >
    <PipingComponent ID="SwingCheckValve-1"
      ComponentClass="SwingCheckValve"
      ComponentClassURI="http://data.posccaesar.org/rdl/RDS444129">
      <Tag>SCV-1A</Tag>
      <ConnectionPoints FlowIn="1" FlowOut="2">
          <Node ID="SwingCheckValve-1-Inlet" />
          <Node ID="SwingCheckValve-1-Outlet" />
      </ConnectionPoints>
    </PipingComponent>
  </PipingNetworkSegment>
</PipingNetworkSystem>
</PlantModel>
```

Instrument Generation — Our Copilot

```xml
<PlantModel>
 <PlantInformation ... >
     <UnitsOfMeasure />
 </PlantInformation>
 <PipingNetworkSystem ... >
   <PipingNetworkSegment ... >
      <Connection FromID="Tank-1" ToID="SwingCheckValve_1"
        FromNode="2" ToNode="1" />
      <CenterLine NumPoints="0" />
      <PipingComponent ID="SwingCheckValve_1"
         ComponentClass="SwingCheckValve"
         ComponentClassURI="http://data.posccaesar.org/rdl/RDS610424">
         <ConnectionPoints FlowIn="1" FlowOut="2" NumPoints="3">
            <Node ID="SwingCheckValve_1-DefaultNode" />
            <Node ID="PipingNode-11" Type="process" />
            <Node ID="PipingNode-12" Type="process" />
         </ConnectionPoints>
      </PipingComponent>
   </PipingNetworkSegment>
 </PipingNetworkSystem>
<PlantModel>
```

Actuating System Generation — Zero-shot

```xml
<PlantModel>
 <PlantInformation ... >
   <UnitsOfMeasure/>
 </PlantInformation>

 <Instrumentation>
    <ActuatedValve ID="100001">
        <Tag>GlobeValve-1-Actuator</Tag>
        <ActuatedValveType>Globe</ActuatedValveType>
        <EquippedWith>
            <Actuator ID="100002">
                <Tag>GlobeValve-1-Actuator</Tag>
                <ActuatorType>Pneumatic</ActuatorType>
            </Actuator>
        </EquippedWith>
    </ActuatedValve>
 </Instrumentation>

</PlantModel>
```

Actuating System Generation — Few-shot

```xml
<InstrumentationLoopFunction ... >
  <Association Type="is a collection including"
    ItemID="ProcessInstrumentationFunction"/>
</InstrumentationLoopFunction>
<ProcessInstrumentationFunction ... >
  <Association Type="is a part of"
     ItemID="InstrumentationLoopFunction"/>
  <Association Type="is logical end of"
     ItemID="MeasuringLineFunction"/>
  <ProcessSignalGeneratingFunction ...>
    <Association Type="is located in" ItemID="Nozzle-2"/>
    <Association Type="is logical start of"
       ItemID="MeasuringLineFunction-2"/>
  </ProcessSignalGeneratingFunction>
  <InformationFlow ... ComponentClass="MeasuringLineFunction" ... >
    <Association Type="has logical start"
       ItemID="ProcessSignalGeneratingFunction-2"/>
    <Association Type="has logical end"
       ItemID="ProcessInstrumentationFunction-2"/>
  </InformationFlow>
</ProcessInstrumentationFunction>
```

* generated XML does not have any ActuatingSystem, ActuatingFunction, SignalLine and Nodes

Actuating System Generation — Our Copilot

```xml
<ActuatingSystem ... >
   <Association Type="fulfills" ItemID="ActuatingFunction_1" />
     <ActuatingSystemComponent ... >
        <ConnectionPoints NumPoints="2">
           <Node ID="DefaultNode" /> <Node Type="signal" ID=... />
        </ConnectionPoints>
      </ActuatingSystemComponent>
      <ActuatingSystemComponent ... >
        <Association Type="refers to" ItemID="GlobeValve_1" />
      </ActuatingSystemComponent>
  </ActuatingSystem>
<ProcessInstrumentationFunction ... >
  <ConnectionPoints NumPoints="1">
     <Node ID="ProcessInstrumentationFunction_1-DefaultNode" />
     <Node Type="signal" ID="InstrumentationNodePosition-2" />
  </ConnectionPoints>
  <Association Type="is logical start of" ItemID="SignalLineFunction_1" />
  <ActuatingFunction ... >
     <Association Type="is fulfilled by" ItemID="ActuatingSystem_1" />
     <Association Type="is logical end of" ItemID="SignalLineFunction_1" />
  </ActuatingFunction>
    <InformationFlow ... ComponentClass="SignalLineFunction" ... >
       ...
       <CenterLine NumPoints="0" />
       <Connection FromID="ProcessInstrumentationFunction_1"
            ToID="ActuatingFunction_1" ... />
     </InformationFlow>
  </ProcessInstrumentationFunction>
```

* generated XML does not have ProcessSignalGeneratingFunction

Figure 3: Illustrative outputs from zero-shot generation, few-shot generation, and our proposed Copilot. Errors are highlighted in red for clarity. Ellipsis indicate correct but lengthy content. The ACPID Copilot exhibits observable improvement in reliability, producing near-ground-truth examples of DEXPI XML.

Method	Completeness
Zero-Shot	0%
Few-Shot	68.28%
ACPID copilot	**92.97%**

Table 2: Comparing soundness during generation, between Zero-shot, Few-shot generation from GPT-4-Turbo and the ACPID copilot.

about DEXPI XML. In contrast, we observe that the ACPID copilot outperforms both vanilla zero-shot and few-shot generation by a significant margin over the small test-bench. We see that few-shot is able to match performance on simpler tasks like addition of Equipments and Attributes, but fails to extend a similar performance on more complicated tasks like addition of Connections, Instruments and Actuating Systems. We can see some places where our copilot makes errors, but the source of such errors can be identified and corrected due to increased provenance of the entire generation process.

The high scores, of the ACPID copilot, in soundness and completeness can be attributed to a more rigid runtime framework because of the rule-based determinism. Additionally, our experiments highlight another advantage: leveraging the Plan and Execute Agents enables our copilot to handle complex prompts—such as incorporating multiple elements and attributes or resolving ambiguous element references—where alternative methods prove inadequate. Finally, our copilot's agentic ability to directly edit the XML eliminates the need to provide the entire current XML as context, as is required in zero-shot and few-shot generation approaches for XML completion, hence saving tokens to partially offset additional tokens required during Planning and Execution. However, there are some limitations which we discuss in the following section.

Discussion and Future Work

Even though the ACPID copilot outperforms the zero-shot and few-shot competitors, due to the rigidity of the rules, the prompts need to be constructed carefully for the entire workflow to run end-to-end. However, this task can be automated in the future using Prompt Automation as depicted in the Figure 2. Additionally, ensuring high accuracy alongside completeness and soundness during generation comes with the trade-off of increased inference time, in comparison to the other methods. However, we estimate that our copilot results in a net reduction in the average time required to create a diagram in comparison to manual generation, by improving efficiency and productivity.

Also, a limitation of our study is the evaluation conducted on a small test bench, primarily due to the scarcity of standardized DEXPI XML data from real-world industrial plants. This limitation arises from the proprietary nature of majority of such data, which restricts its availability for research and validation purposes.

Finally, further enhancements could be made to our Copilot to make it more optimized with respect to the inference time and number of generated tokens. Work can also be done to expand the current capabilities of the copilot to support other engineering diagrams, to improve the natural language processing aspects for better understanding of complex descriptions, and to integrate additional feedback mechanisms for continuous learning.

Conclusion

This acts as a preliminary work in completely automating P&ID generation. In this work, we have presented a novel approach to generate P&IDs directly from natural language descriptions, addressing a significant gap in the existing literature. Our proposed copilot leverages a multi-step agentic workflow to facilitate subsystem-level generation, enhancing the efficiency and accuracy of P&ID creation. To the best of our knowledge, ours is the first Generative AI based Designer Agent (Batres, Lu, and Naka 1997) for P&ID generation. By incorporating Generative AI, we aim to alleviate the tedious and error-prone nature of manual diagram generation, ultimately streamlining the design process in the chemical and process industries.

Looking ahead, our research opens several avenues for future exploration. Our research also opens new avenues to create structured diagrams like architectural blueprints and circuit-boards by adopting our novel workflow, especially in data-constrained environments where knowledge from pre-trained models can be exploited. As Generative AI continues to evolve, we anticipate that our work will contribute to a more efficient and intelligent design process ultimately benefiting all stakeholders involved.

References

Batres, R.; Lu, M. L.; and Naka, Y. 1997. An agent-based environment for operational design. *Computers & Chemical Engineering*, 21: S71–S76. Supplement to Computers and Chemical Engineering.

DEXPI Consortium. 2020a. DEXPI P&ID Specification 1.3. Accessed: 2024-10-25.

DEXPI Consortium. 2020b. First Exemplary Process and Instrumentation Diagram for the Application of Data Exchange. [Figure; Accessed: October 25, 2024].

DEXPI Consortium. 2022. Example P&IDs for DEXPI 1.3. https://gitlab.com/dexpi/TrainingTestCases/-/tree/master/dexpi%201.3?ref_type=heads.

Gozalo-Brizuela, R.; and Garrido-Merchán, E. C. 2023. A survey of Generative AI Applications. arXiv:2306.02781.

Hirtreiter, E.; Schulze Balhorn, L.; and Schweidtmann, A. M. 2023. Toward automatic generation of control structures for process flow diagrams with large language models. *AIChE Journal*, 70(1).

Hirtreiter, E.; Schulze Balhorn, L.; and Schweidtmann, A. M. 2024. Toward automatic generation of control structures for process flow diagrams with large language models. *AIChE Journal*, 70(1): e18259.

International Organization for Standardization. 2022. ISO/IEC 15962:2022 - Information technology — Radio frequency identification (RFID) for item management —

Data protocol: data encoding rules and logical memory functions. Accessed: 2024-10-25.

Kim, B. C.; Kim, H.; Moon, Y.; Lee, G.; and Mun, D. 2022. End-to-end digitization of image format piping and instrumentation diagrams at an industrially applicable level. *Journal of Computational Design and Engineering*, 9(4): 1298–1326.

Koziolek, H.; and Koziolek, A. 2023. LLM-based Control Code Generation using Image Recognition. *ArXiv*, abs/2311.10401.

Laskar, M. T. R.; Bari, M. S.; Rahman, M.; Bhuiyan, M. A. H.; Joty, S.; and Huang, J. 2023. A Systematic Study and Comprehensive Evaluation of ChatGPT on Benchmark Datasets. In Rogers, A.; Boyd-Graber, J.; and Okazaki, N., eds., *Findings of the Association for Computational Linguistics: ACL 2023*, 431–469. Toronto, Canada: Association for Computational Linguistics.

Levy, M.; Jacoby, A.; and Goldberg, Y. 2024. Same Task, More Tokens: the Impact of Input Length on the Reasoning Performance of Large Language Models. In Ku, L.-W.; Martins, A.; and Srikumar, V., eds., *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 15339–15353. Bangkok, Thailand: Association for Computational Linguistics.

Nasby, G. 2012. Using Process Flowsheets as Communication Tools. *Chemical Engineering Progress*, 108: 36–44.

Paliwal, S.; Jain, A.; Sharma, M.; and Vig, L. 2021. Digitize-PID: Automatic Digitization of Piping and Instrumentation Diagrams. In Gupta, M.; and Ramakrishnan, G., eds., *Trends and Applications in Knowledge Discovery and Data Mining*, 168–180. Cham: Springer International Publishing. ISBN 978-3-030-75015-2.

ProteusXML. 2024. Proteus 4.1 Schema. Accessed: 2024-10-25.

Rich, S. H.; and Venkatasubramanian, V. 1987. Model-based reasoning in diagnostic expert systems for chemical process plants. *Computers & Chemical Engineering*, 11(2): 111–122.

Sakhinana, S. S.; Sannidhi, G.; and Runkana, V. 2024. Towards Human-Level Understanding of Complex Process Engineering Schematics: A Pedagogical, Introspective Multi-Agent Framework for Open-Domain Question Answering. arXiv:2409.00082.

Siirola, J. J.; and Rudd, D. F. 1971. Computer-Aided Synthesis of Chemical Process Designs. From Reaction Path Data to the Process Task Network. *Industrial & Engineering Chemistry Fundamentals*, 10(3): 353–362.

Stephanopoulos, G.; Henning, G.; and Leone, H. 1990. MODEL.LA. A modeling language for process engineering—I. The formal framework. *Computers & Chemical Engineering*, 14(8): 813–846.

Stephanopoulos, G.; Johnston, J.; Kriticos, T.; Lakshmanan, R.; Mavrovouniotis, M.; and Siletti, C. 1987. Design-kit: An object-oriented environment for process engineering. *Computers & Chemical Engineering*, 11(6): 655–674.

Venkatasubramanian, V. 2019. The promise of artificial intelligence in chemical engineering: Is it here, finally? *AIChE Journal*, 65(2): 466–478.

Vogel, G.; Schulze Balhorn, L.; and Schweidtmann, A. M. 2023. Learning from flowsheets: A generative transformer model for autocompletion of flowsheets. *Computers & Chemical Engineering*, 171: 108162.

Wang, L.; Xu, W.; Lan, Y.; Hu, Z.; Lan, Y.; Lee, R. K.-W.; and Lim, E.-P. 2023. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. In *Annual Meeting of the Association for Computational Linguistics*.

Wiedau, M.; von Wedel, L.; Temmen, H.; Welke, R.; and Papakonstantinou, N. 2019. ENPRO Data Integration: Extending DEXPI Towards the Asset Lifecycle. *Chemie Ingenieur Technik*, 91(3): 240–255.

YM, P.; Ganesan, V.; Arumugam, D. K.; Gupta, M.; Shadagopan, N.; Dixit, T.; Segal, S.; Kumar, P.; Jain, M.; and Rajamani, S. 2023. PwR: Exploring the Role of Representations in Conversational Programming.

Zhang, J.; Huang, J.; Jin, S.; and Lu, S. 2024. Vision-Language Models for Vision Tasks: A Survey. arXiv:2304.00685.