# Co-pilots for Arrowhead-based Cyber-Physical System of Systems Engineering

Csaba Hegedűs, Pál Varga
Department of Telecommunications and Artificial Intelligence
Budapest University of Technology and Economics
Műegyetem rkp. 3., H-1111 Budapest, Hungary.
Email: {hegeduscs, pvarga}@tmit.bme.hu

***Abstract*—One benefit of Large Language Model (LLM) based applications (e.g. chat assistants or co-pilots) is that they can bring humans closer to the loop in various IT and OT solutions. Co-pilots can achieve many things at once, i.e. provide a context-aware natural language interface to knowledge bases, reach various systems (via APIs), or even help solving multi-step problems with their planning and reasoning abilities. However, making production-grade chat assistants is a topical challenge, as fast-evolving LLMs expose new types of application design and security issues that need tackling. These especially rise to power when we try to apply these solutions to industrial automation use cases, as they need additional explainability and reliability engineered into the architecture. This paper describes the envisioned use cases and the findings of proof of concept copilots for the Cyber-Physical System of Systems (CPSoS) engineering domain. The paper suggests three types of copilots to support the stages throughout the CPSoS engineering lifecycle – and shows Proof-of-Concept scenarios for the Eclipse Arrowhead engineering process.**

## I. Introduction

The year 2023 has mostly been spent around generative Artificial Intelligence (genAI) and Large Language Models (LLM). Various commercial products and open-source projects have appeared and are showing great evolution in just a very short timeframe. Most notable ones include the products of OpenAI and Microsoft, ChatGPT 4 and its text-to-image model Dall-E [1].

Now, every industry and domain is looking at how to utilize these technologies in their business and keep up with their development speed in the productization phases. Building solutions with these products capable of semantic reasoning and contextual content generation, among others, enables a lot of new ideas to come forth from product owners [2]. Many companies, Microsoft for example, are building their LLM- and genAI-based product portfolio around the concept of *co-pilots*.

Co-pilots, or chat assistants, refer to an AI-based solution aiding users in various customer journeys of an ecosystem. For example, the Microsoft Copilot [3] suite is a horizontal solution in the Microsoft 365 and Azure ecosystems that has access to the user's (and organization's) data in Azure and Entra ID , and can interact with various Microsoft 365 (Office, Power and Azure platform) applications via Application Programming Interfaces (APIs) and specialized Domain Specific Languages (DSLs) to generate, summarise or augment (multi-modal) content; or act on behalf of the user within these applications.

Nevertheless, rightful concerns have arisen around the safety, reliability and the *responsible use* of all AI-based solutions. Explainable AI (XAI) emphasizes transparent model architectures and feature importance rankings, as it is essential to comprehend the reasoning behind the AI's decisions and actions. This is especially the case for potential industrial (manufacturing) applications, as ours. Our chat assistant needs to be able to provide plans and explanations that consider the specific industrial domain knowledge, so its users can trust and validate the outcome [4].

### A. Capabilities of Chat Assistants

Based on the current state of the business, these chat-based co-pilot agents can have many interesting features. Primarily, the capabilities depicted in Table I are considered for this paper. Besides these, there is one additional capability worth mentioning: the code interpreter sandbox, where the model can run generated code snippets against uploaded files. However, this feature is out of the scope of this paper.

Most of these functionalities can be developed using modern, cloud-native application design patterns and LLM application orchestration platforms, such as LangChain [5], llamaindex [6], Haystack [7] or Semantic Kernel and Memory (from Microsoft) [8].

### B. Motivation and Structure of the Paper

The purpose of this paper is to demonstrate that *LLM-based co-pilots can be implemented for industrial automation as well*, to aid the development and operations of large-scale System of Systems (SoS). We will also report on the qualitative findings around a proof-of-concept (PoC) demonstrator that integrates into the framework and showcases the values added across the Arrowhead toolchain.

This work is to be considered in the context of the Eclipse Arrowhead framework [9], which is a Service Oriented Architecture (SOA) driven design framework and middleware implementation to govern and orchestrate Cyber-Physical System of systems (CPSoS). Figure 1 depicts the deployment structure

TABLE I
CAPABILITIES OF CHAT ASSISTANTS

| Capability | Description | Underlying technique |
|---|---|---|
| Chatbot persona | Talk with a specified persona over chat with conversational safeguards | Persona prompting, few-shot examples, model finetuning, conversation starter templates |
| Knowledge-based chat | Talk to the chatbot that can access and understand multimodal documents | Retrieval Augmented Generation (RAG) and model finetuning |
| Chat memory | Chatbot can remember earlier discussions | Context serialized into prompt (part of grounding) and RAG |
| Semantic planning | Chatbot able to generate train-of-thought and action plans towards a specific target | Prompt (and semantic skills) engineering, intent analysis |
| External integrations | Chatbot is able to execute API calls to targets | Copilot orchestration middleware and openAPI or DSL-based plugins, fine-tuning with application DSLs |
| Web search | Chatbot can look up websites using search engines | Search query semantic skills and web crawler middleware |
| Integration of Copilot | Chatbot can be integrated into websites, messaging cross-platforms and desktop applications | API web services and cross-platform frontends for the copilot |

of an *Arrowhead Local Cloud*, where the colored objects are the Core Systems of the framework (Service Registry, Orchestrator and Authorization) that govern the rest of the networked Cyber-Physical Systems, abstracted as grey-boxed *Application Systems*, with a device abstraction underneath as well. The *Arrowhead Management Tool* supports engineers to configure and provide dashboards for the contents of the Core Systems, which uses dedicated, internal REST APIs of the Core Systems. The Arrowhead Local Cloud is an on-prem architected collection of microservices that do not have services exposed over the Internet [10], and which communicate via a detached Public Key Infrastructure, using mutual Transport Layer Security (mTLS) authentication [11].

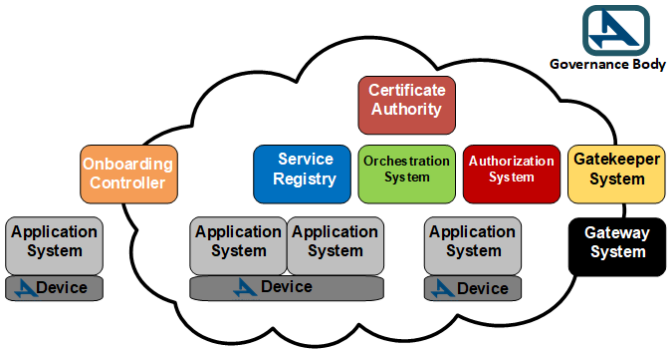


Fig. 1. A generic overview on an Arrowhead local cloud [10]

Chapter II discusses the technological state of the art (SotA) of Table I. Chapter III details the implemented an Arrowhead-based proof-of-concept (PoC) demonstrator application and presents its findings and future work, while Chapter IV concludes the paper.

## II. RELATED WORKS

### A. LLM Engineering Used in Chat Assistants

While classic chatbot engineering already had custom topics, workflows and skills, their user journeys were mostly scripted, leaving minimal reactive dynamism in their use. This all changes now with LLMs, as they are capable of content and reasoning generation on a much higher level. With proper prompt engineering and supporting LLM orchestration middleware, we can build applications that make semantic decisions based on a knowledge base and various real-time integrations as well.

In general, LLMs take user prompts (of a maximum size) and generate responses to them. With ChatGPT 4 [1] and Google Gemini [12], the input prompt can even be multi-modal, i.e. feature image or video files with obvious restrictions (e.g. on length or size). The chat message prompts are typically structured into an array of messages annotated with either a (i) system, (ii) user, (iii) assistant or (iv) other role. Developers can specify *system prompts* that are always pre-appended to user prompts, and enable the developer to specify the assistant's persona and provide other general instructions for the assistant responses. This system part needs to be carefully (prompt-)engineered to restrict the behavior of the LLM. System prompts can also be used to specify many things, e.g. enforce what syntax highlights to be used in the responses or to differentiate between e.g. generated code snippets, markup documents or citations from plain text.

User prompts must be pre-processed before being submitted to the LLM. Besides checking it against various policies and safeguards (e.g. foul language), security scans (e.g. injections or *jailbreaks*) [13], chat copilots need to understand the actual intent behind the prompt (i.e. intent analysis) and then provide the necessary *groundings* for the LLM to accomplish the user request. Groundings refer to all contextual details added by the copilot engine to the prompt, such as user details, relevant chat history, or even user rights (i.e. potential callable services). This can help avoiding hallucinations and provide the LLM with guardrails in its reasoning. When attempting to establish whether or not something in an LLM response is 'true', we can either check for it in the supplied prompt (this is called 'narrow grounding') or use some general knowledge ('broad grounding') [8].

Another technique we use is called Retrieval Augmented Generation (RAG) [14]. This is also a form of grounding, but based on a knowledge base we built. This may be generated from multimodal documents where the content is processed by an *embedding model* and stored as a vector in a vector

database [15], as depicted in Fig. 2. During user interactions, the co-pilot middleware queries the vector database to find the most relevant snippet(s) of the knowledge base, based on the semantic embedding and the understanding of the intent from the user prompt, and adds these snippets the final prompt. Afterwards, based on these injected groundings, the LLM can make a more educated answer. RAG is an active research area, where the ability to properly reason depends on numerous factors [16], [17]. There is no single implementation of a RAG solution that can work well across the board, since the design of the data pipeline, the source materials, the embedding process and the retrieval process itself all affect its dependability.

In case very large ready-made materials, knowledge base and Q&A are available for use in a specific task of the co-pilot, we could also *finetune* our LLM for the purpose, but this technique is not considered in this work.

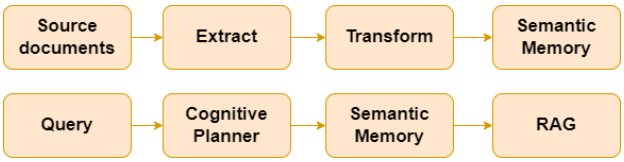


Fig. 2. Overview of Retrieval Augmented Generation

LLMs are great at working with Domain Specific Languages (DSL). Out-of-the-box, or based on few-shot examples, LLMs can correctly understand and then generate syntactically and semantically correct DSL artifacts. These include openAPI specifications [18] as well. This allows for external integrations, i.e. allows the LLM to interact with public or private RESTful APIs and hence act on behalf of the end user (using e.g. oAuth2 authentication). This is enabled by the co-pilot middleware solution, where the middleware is able to (i) serve the LLM the potential operations from the openAPI manifest, (ii) inject the details of an operation selected,(iii) execute the API call specified by the LLM, (iv) serve the LLM the response from the API call(s).

These functions are called *plugins* and, together with other *semantic skills*, form the core of co-pilot applications. Semantic skills are prompt templates that can be filled up by the co-pilot middleware to get any AI model to execute specific tasks (e.g. solve math problems correctly or tell jokes). Plugins and skills have a prompt template that declares their input and output, e.g.:

```
[AVAILABLE FUNCTIONS]
Name: 'MathPlugin-Ad'
Description: 'Adds two numbers'
Inputs:
-number1:double-'The first number to add'
-number2:double-'The second number to add'
Outputs:
-number3:double-'Sum of the input'
```

In chat assistants, the main goal is to create chatbots that can understand complex tasks, plan and execute appropriate workflows on their own, and understand the responses from external services, while all based on the natural language chat with its users. This is essentially the application of plan-and-solve prompting, but allowing the assistant to acquire additional, external information if needed, or act on behalf of the user. The assistant can essentially propose an execution plan (course-to-fine) and then act upon it. There are already existing prompting patterns available for planners in LLMs, such as Sequential Planners [8], ReAct [19], Plan-and-Solve [20], RestGPT[21] or Reasoning-via-Planning [22].

Finally, the User Experience (UX) of these assistants is just as essential, since the user shall be able to view and potentially modify the plans proposed by the assistant and follow the plugin calls made on behalf of the user as well. In case of API calls, the user should have the ability to review and modify the requests to-be-sent in the plan. It is also essential to allow for telemetry collection, i.e. users scoring the responses and actions of the assistant. The telemetry can later on be reused for finetuning as part of the *LLM-ML Ops* pipelines.

### B. Copilot Products on the Market

There are numerous commercial products on the market already, capable of some of the features mentioned in the previous sections. Most notable ones are OpenAI's ChatGPT Assistant [1], Google Bard [12] and Microsoft Copilot Studio [23]. These are public Software-as-a-Service offerings, and offer no-code platforms for developing chat assistants. However, these cutting-edge services have some shortcomings for our specific industrial use case.

- Underlying solutions are rapidly evolving with breaking changes, as the platforms mature.
- Can only integrate to public API services over the Internet, signed by a global Certificate Authority. They do not support mTLS.
- They don't allow customization of the underlying services and capabilities, such as RAG pipeline or planner.
- They currently do not allow users to customize UX.
- Such SaaS solutions generally cannot be deployed on-prem.

## III. Copilots Across the SoS Lifecycle

### A. Use Cases for Arrowhead Copilots

Based on this, we can establish our use cases for co-pilots in the Arrowhead Engineering Process (AEP) [24] for CPSoS. The mission is *the ability to design and manage a CPSoS via natural language and analyze its status and monitoring data using the potentially multi-modal output of the Copilot(s)*. We can conceptualize three stages of Copilots in AEP (see Fig. 3), with increasing complexity and value-added:

1) An *Arrowhead Expert* providing a chat-based entry point into the Arrowhead architecture, feeding on the already existing documentation and publications around

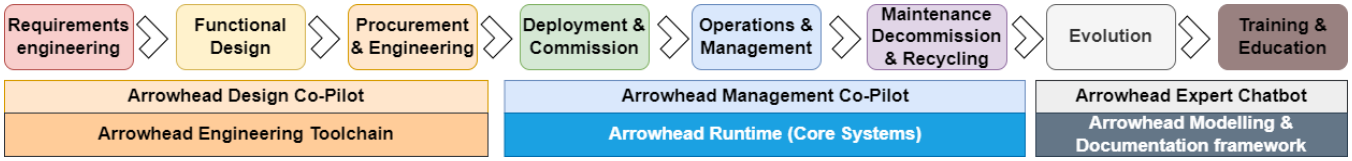


Fig. 3. The graphical overview of the Arrowhead Engineering Process (AEP) [24] to be supported by Co-pilots

the ecosystem, answering design and integration-related questions. This can be embedded in the Arrowhead Framework Wiki [9] as an inline chatbot. Intended users are anyone who visits the Wiki.

2) The *Arrowhead Management Copilot* interacting with various Arrowhead Core Systems of a Local Cloud deployment to analyze and understand, potentially manage the CPSoS via the Arrowhead governing middleware. This tool can be embedded as a widget to the Arrowhead Management Tool GUI. Intended users are the authenticated Local Cloud (SoS) operators.
3) *Arrowhead Design Copilot* which can integrate with the engineering toolchain to design SoS deployment and underlying industrial automation processes and infrastructure (i.e. SysML modeling with Eclipse Papyrus). This co-pilot can be integrated into the Arrowhead Engineering Toolchain, interacting with the design tools for CPSoS. The intended users are the SoS engineers.

This work demonstrates the first and second use cases, and these require fairly different setups and complexity. The Arrowhead Expert is primarily a RAG-oriented Q&A assistant, without the need to use long-term memory. Fig. 4 depicts the RAG solution components, as demonstrator uses Microsoft Azure resources for persistence, and relies on the Semantic Kernel middleware. Its primary knowledge base comes from the Arrowhead documentation and research papers (in IEEE format). We have also provided a few-shot training on sample question-answers, but we primarily rely on the persona prompting pattern to define how the Expert shall conduct itself in general:

*This is a chat between an intelligent chatbot named Arrowhead Expert and one human participant. The Arrowhead Expert is an expert in the Arrowhead framework and can access Arrowhead documentation and publications. The user would like to use Arrowhead for engineering (CPSoS), so help them learn and adapt the Arrowhead framework. Try to be concise with your answers, and rely only on the documentation snippets provided below. If you don't know the answer, don't make things up, but say so. Do not answer questions outside of the Arrowhead context.*

This expert is able to answer reasonable questions like *"What are the restrictions on system naming in Arrowhead?"*, or *"How can manually generate certificates for a new application system, if I do not have Onboarding Controller?"*. It will not remember users after the session, or their chat history (only up until chaining previous messages of the session can

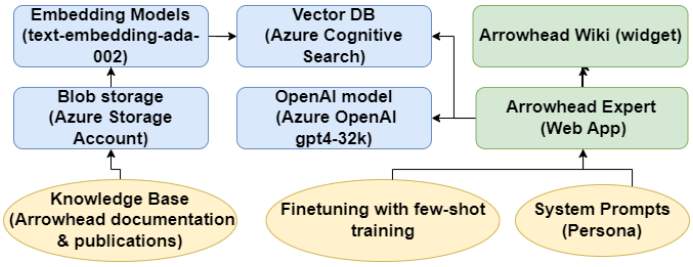


Fig. 4. Overview of the Arrowhead Expert

fit into the prompt as context), but it is also not necessary. However, it can provide the relevant *references* to the documentation, where the answer can be further researched. Its initial knowledge base consists of Arrowhead publications, API and developer documentation of the core systems, and the contents of the Arrowhead Wiki, altogether 42 documents for the PoC.

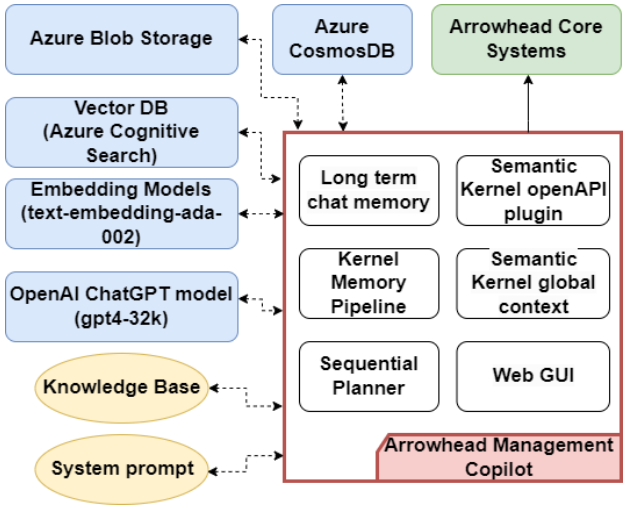


Fig. 5. Overview of the Arrowhead Management Copilot

Meanwhile, the Arrowhead Management Co-pilot is a more complex assistant, as depicted in Fig. 5. It needs to (i) be able to plan, (ii) remember chat context for better planning, and (iii) execute API calls against the Arrowhead Core Systems. This application is deployed with Azure private resources, since proper AAA is missing yet in the implementation. The vector database, embedding model and blob storage are needed for storing long-term chat and document-based memory (and enable RAG on top of them), while CosmosDB stores the chat interactions and the plans proposed by the bot with other

configuration items (like persona prompts, etc.). The solution is deployed in the same private network as the Arrowhead core systems. The persona of this co-pilot is phrased as follows:

*This is a chat between an intelligent assistant named Arrowhead Co-pilot and a human participant. The Arrowhead Co-pilot is an expert in Arrowhead framework and can access the Arrowhead core systems via API plugins. Try to be concise with your answers, and primarily rely on the Arrowhead core systems for providing up-to-date data and configuration from the system of systems. If you don't know the answer or don't know how you can access the requested information, don't make things up, but say so. You can also ask follow-up questions for clarification on what is required by the user.*

This assistant is able to answer simple questions like *"What are the currently registered services in the Arrowhead Service Registry?"*, but it can also chain API calls together in sequential plans, if we require more complicated tasks, such as *"Please add an authorization rule between systems P and C, for service S in the Arrowhead intra-cloud Authorization System, but please register system X as the provider of service S in the Arrowhead Service Registry first"*.

### B. Findings of the Proof of Concept

These PoC applications showcase good initial impression. Validation of the PoC has been done in a small workshop with potential stakeholders. Both assistants can handle Arrowhead-context and act as their appropriate personas with the intended guardrails. The document RAG, long-term memory, plugin handling (i.e. making API calls to Arrowhead) and basic planner features are well received. The general UX, however, is inadequate to our industrial purposes and response times are considered long.

The document RAG, with the current semantic memory capabilities, does not allow for utilizing the multi-modality of (mostly IEEE-formatted) scientific publications and Arrowhead documentation. This RAG engine, while works acceptably acting as long-term chat memory, is unable to work with tables or diagrams of papers, which is a great loss of semantic content. This renders the Expert unable to help with e.g. the Arrowhead SoS modeling concepts and visualization tasks. Moreover, the somewhat "naive" document chunking strategy (i.e., every 1000 characters) is seriously limiting usability, as words are often even cut in half. Another related problem, albeit humorous, is that publications often contain (accidental) hyphenations, which causes problems in understanding technical specifications (i.e. in certificate CN names: *arrowhead.eu* vs. *arrow-head.eu*).

Connecting the LLMs with the Arrowhead APIs also brings practical challenges. Plugins utilize openAPI3.0 specifications, i.e. they work with the operationId and description of each endpoint first, then with the path templates, headers, request and response structures. It apparently does not bode well with the LLM, if the openAPI specification is "too" long (even if it fits the token limit) or contains a lot of seemingly similar endpoints. It can also cause unfortunate "misunderstandings" if path templates or JSON objects have depth (e.g. over 3) to them, or if the openAPI document describes them in an object-oriented manner (e.g. using AllOf, AnyOf directives). An additional problem is when whole JSON objects or values can still be hallucinated, even if the prompt seems well grounded. OpenAI has published a guideline on plugin openAPI specifications, but this could warrant further research [25].

Moreover, the assistant is able to put together API calls from different plugins as well (i.e. register services with the Service Registry first, then use the acquired IDs to register related authorization rules in Authorization System) in the plans. However, it is not able to handle making the logical connections between the same IDs of Arrowhead objects if those are served via different plugins. This assumes that the LLM is only able to contextualize plugins individually, and hence requires plugin API designs that are self-sufficient. It is also clear that plans are static and the co-pilot has no chance of making adjustments after subsequent API calls are made.

Regarding XAI, the current UI design allows for a minimal view to the behind-the-scenes. Nevertheless, the visualization of the technical prompts, plans and document citations in responses need extra consideration. Moreover, the assistant should be able to respond to follow-up questions on previous plans and results, but the short-term memory cannot produce well-structured output (missing grounding for understanding previously generated internal data structures in the follow-up prompt).

### C. Future Work

There is significant design, implementation and research work lying ahead to mature these co-pilots. The Arrowhead Expert shall be deployed with cloud-native resources, but needs to implement rate limiting and security hardening (e.g. against prompt injections) on its interactions to avoid anonymous jailbreaks. It also shall feature industrial-grade AAA (authentication-authorisation-accounting), while integrating into the Arrowhead PKI.

Meanwhile, the final implementation needs to move away from Azure dependencies, i.e. it needs to support on-prem deployment models and enable the choice of upstream services. It also needs to allow for small on-prem and cloud-native deployments, and will be implemented as an API First headless architecture.

Our RAG pipeline needs a complete design, where it needs to be multi-modal and to process markup documents better (Arrowhead documentation) and research publications (i.e. IEEE formatted). The focus must be put on teaching the Expert about the Arrowhead's UML-based modeling paradigm as well. Regarding the test automation framework for RAG, we need to establish a validation Q&A dataset with Arrowhead context, perhaps from real-life interactions between users and the PoC. Editors of the knowledge base also need automated mechanisms to be able to review the chunking process and

update documents, if needed. Afterwards, the plan is to build a prompt flow-based test automation framework, where answers to test scenarios are to be evaluated by another customized GPT, and act as a quality gate in our knowledge base-related CICD (Cont. Integration and Delivery pipelines).

The Expert and Management Co-pilot needs to become multi-modal as well, i.e. able to handle and use visualizations in their interactions. Therefore, generating tables out of plan results, visualizing graphs and UML diagrams from the Arrowhead runtime are on the backlog. Other XAI-related UX changes will be implemented as well, where users shall be able to track and modify plans, and review and edit raw prompts and API calls made by the plugins to upstream services. User feedback and telemetry collection will allow for collecting potential future test scenarios. Reinforcement learning is only a long-term opportunity.

The planner engine requires work, and should be based on more advanced prompt engineering techniques, like ReAct [19] or interactive planner [26]. After each step, the user and the LLM should have the ability to update the next API calls to be made, based on the results from previous calls. The planner engine could also utilize few-shot learning, as indicated in LLM planner surveys [27].

Finally, the Arrowhead framework itself needs additional work as well to integrate better with co-pilots. Each core service requires additional endpoints and business logic to serve compliant to the openAI plugin specifications [25]. Moreover, the individual API designs need to adapt and feature better segregation of duties, so LLMs can understand available methods and their templates better.

## IV. Conclusions

The design of LLM-based applications, like chat assistants, is a topic of day-by-day rapid evolution. From idea formation to productizing the solution, there are no existing industrial best practices or really reusable frameworks yet. This paper describes the first steps in the process of creating chat co-pilots for the CPSoS engineering domain, starting by analyzing achievable capabilities offered by LLM SotA, establishing feasible use cases for the technology, creating a proof-of-concept implementation; up to specifying an R&D roadmap, all in the context of the Arrowhead framework.

## Acknowledgment

Project No. KDP-IKT-2023-900-I1-00000957/0000003 has been implemented with the support provided by the Ministry of Culture and Innovation of Hungary from the National Research, Development and Innovation Fund, financed by the Co-operative Doctoral Program [C2299296] funding scheme.

Part of this work is created within the *Arrowhead fPVN* project, supported by the Chips-JU under grant agreement no. 101111977.

## References

[1] OpenAI, "Chatgpt: Optimizing language models for dialogue," https://www.openai.com/chatgpt, 2023, acc: 2023-12-28.

[2] C. Parnin, G. Soares, R. Pandita, S. Gulwani, J. Rich, and A. Z. Henley, "Building your own product copilot: Challenges, opportunities, and needs," 2023.

[3] Microsoft, "365 Copilot," https://learn.microsoft.com/en-us/microsoft-365-copilot/microsoft-365-copilot-overview, 2023, acc: 2023-12-28.

[4] P. J. Phillips, C. A. Hahn, P. C. Fontana, D. A. Broniatowski, and M. A. Przybocki, "Four principles of explainable artificial intelligence," *NISTIR, Gaithersburg, Maryland*, vol. 18, 2020.

[5] LangChain, "LLM Orchestration," https://www.langchain.com/, 2023, acc: 2023-12-28.

[6] Llamaindex, "Data framework for llm applications," https://www.llamaindex.ai/, 2023, acc: 2023-12-28.

[7] DeepSet AI, "Haystack," https://github.com/deepset-ai/haystack, 2023, acc: 2023-12-28.

[8] Microsoft, "Semantic kernel," https://github.com/microsoft/semantic-kernel, 2023, acc: 2023-12-28.

[9] Eclipse Foundation, "The Arrowhead Framework," https://projects.eclipse.org/projects/iot.arrowhead.

[10] C. Hegedus, P. Varga, and S. Tanyi, "A governance model for local and interconnecting arrowhead clouds," 10 2020.

[11] S. Plósz, C. Hegedus, and P. Varga, "Advanced security considerations in the arrowhead framework," vol. 9923, 09 2016, pp. 234–245.

[12] Google, "Bard," https://bard.google.com/, 08 2023.

[13] OWASP, "Top 10 for LLM Applications," https://llmtop10.com/, acc: 2023-12-28.

[14] S. Siriwardhana, R. Weerasekera, E. Wen, T. Kaluarachchi, R. Rana, and S. Nanayakkara, "Improving the Domain Adaptation of Retrieval Augmented Generation (RAG) Models for Open Domain Question Answering," *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 1–17, 01 2023.

[15] I. Ilin, "Advanced RAG Techniques: an Illustrated Overview," https://pub.towardsai.net/advanced-rag-techniques-an-illustrated-overview-04d193d8fec6.

[16] P. BehnamGhader, S. Miret, and S. Reddy, "Can retriever-augmented language models reason? the blame game between the retriever and the language model," 2023.

[17] Z. Shao, Y. Gong, Y. Shen, M. Huang, N. Duan, and W. Chen, "Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy," 2023.

[18] OpenAPI Initiative, "The openapi specification," https://www.openapis.org/, acc: 2023-12-28.

[19] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," 2023.

[20] L. Wang, W. Xu, Y. Lan, Z. Hu, Y. Lan, R. K.-W. Lee, and E.-P. Lim, "Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models," 2023.

[21] Y. Song, W. Xiong, D. Zhu, W. Wu, H. Qian, M. Song, H. Huang, C. Li, K. Wang, R. Yao, Y. Tian, and S. Li, "Restgpt: Connecting large language models with real-world restful apis," 2023.

[22] S. Hao, Y. Gu, H. Ma, J. J. Hong, Z. Wang, D. Z. Wang, and Z. Hu, "Reasoning with language model is planning with world model," 2023.

[23] Microsoft, "Copilot Studio," https://www.microsoft.com/en-us/microsoft-copilot/microsoft-copilot-studio, 2023, acc: 2023-12-28.

[24] G. Kulcsár, P. Varga, M. S. Tatara, F. Montori, M. A. Inigo, G. Urgese, and P. Azzoni, "Modeling an industrial revolution: How to manage large-scale, complex iot ecosystems?" in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2021.

[25] OpenAI, "ChatGPT Plugins : Getting Started," https://platform.openai.com/docs/plugins/getting-started, 2023, acc: 2023-12-28.

[26] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su, "Llm-planner: Few-shot grounded planning for embodied agents with large language models," 2023.

[27] S. Qiao, Y. Ou, N. Zhang, X. Chen, Y. Yao, S. Deng, C. Tan, F. Huang, and H. Chen, "Reasoning with language model prompting: A survey," 2023.