Enterprise Modelling and Information Systems Architectures
Vol. 20, No. 3 (2025). DOI:10.18417/emisa.20.3

LLM, LAM, LxM Agent: From Talking to Acting Machines

Insights from the Perspective of Conceptual Modeling

Peter Fettke[*,a], Hans-Georg Fill[b], Julius Köpke[c]

[a] Saarland University and German Research Center for Artificial Intelligence (DFKI), Germany
[b] University of Fribourg, Switzerland
[c] University of Klagenfurt, Austria

1 Introduction

1.1 The advent of Large Language Models

Understanding and generating natural language has been a lively field of research in informatics for decades and has received considerable attention in practice since the prominent breakthrough of OpenAI with ChatGPT (Jurafsky and Martin 2025; Zhao et al. 2025). Such systems regularly work on so-called *artificial neural networks*, also known as *sub-symbolic* methods (Smolensky 1990). Technically, this idea has been known for decades, but only recently have new ideas for technical development and training been created; these are known as 'generative pre-trained transformers', or GPT for short (Brown et al. 2020; Radford et al. 2019). This technical approach is generally regarded as the basis for so-called *large language models* (LLM).

An LLM is large in terms of various aspects:

- the number of parameters used by the model,
- the required time for model training, and
- the amount of the training texts and other data sources are impressive.

An LLM can now be used productively for many applications. Typical examples are:

- Writing texts on given topics,
- revising the language and style of existing texts,
- translating texts,
- summarizing texts,
- extracting key statements from texts,
- conducting dialogues in natural language,
- classifying documents,
- answering general questions, and
- identifying positive or negative moods in texts.

Technically, an LLM is based on sequence of *tokens*. A token typically represents a character, a sequence of characters, a word, or longer text fragments. It has been well known for decades, that human-generated texts entail typical statistical patterns (Shannon 1948; Weaver 1949); if certain characters follow each other regularly, such sequences are produced more frequently by an LLM. Nevertheless, it is surprising that an LLM can exhibit such a wide range of capabilities.

1.2 From LLM to LxM agents

For some time, the idea arose that an LLM do not only use tokens for representing natural language but other types of objects. With appropriate adaptions, an LLM can be used for generating program code, protein predictions, image analyses and, more recently, for generating plans, actions, and workflows (Ding et al. 2023; Yenduri et al. 2024). It is astonishing that the same techniques used to generate natural language texts can also be effectively employed to generate other types of objects.

The tokens generated by an LLM which is trained on workflows, actions, processes can directly be used to trigger a (remote) procedure call,

* Corresponding author.
E-mail. peter.fettke@dfki.de

a web service, or a human action. Such a combination is called *large action model* (Wang et al. 2024; J. Zhang et al. 2024), *large process model* (Fettke 2024; Kampik et al. 2023), or *LxM* (Senaratna 2025; Wahlster 2024) – where the letter "x" is used as a placeholder for referring to different concepts.

More recently, LLM are combined with the well-known idea of a *software agent*. Software agents are a long-established concept in informatics, particularly in artificial intelligence. Such an agent has sensors and actuators that enable it to determine information about its environment and initiate or execute actions in the environment. From a broader perspective, a human person can also be understood as an agent.

While an LLM can be understood as a *talking machine*, a software agent equipped with an LLM is an *acting machine*; in short, it is an *LxM agent*. Initial applications exist for the following scenarios, for example:

- 'Buy a black desk on the Internet';
- 'Book a train journey from Saarbrücken to Klagenfurt on the Internet';
- 'Enter the booking document in the ERP system'.

To realize these scenarios or more complicated ones, an LxM agent does not only incorporate an LLM but has other modules for data storage, planning or inference, similar to other intelligent systems. This shows that a powerful LxM agent is not only based on the sub-symbolic concepts of large language models, but can also represent and process explicit, symbolically represented knowledge. For example, legal regulations should rather be modeled explicitly. This is why we also speak of a so-called *hybrid* or *neuro-symbolic* LxM agent.

One further aspect is of importance for understanding the new concept. Historically, the term "agent" is understood as a simple predicate, e.g., a thing x is an agent or not an agent. Recently, more often the characteristic *agentic* is used: A thing x is more or less agentic than a thing y; some things are not agentic at all, for example, a stone or a mountain. Some things are fully agentic, e.g., a human person. In between, a broad spectrum of different agentic characteristics are distinguished (Fettke and Strohmeier 2022; Kapoor et al. 2024):

- Environment and goal: single tasks vs. complex processes;
- User interfaces: technical user interfaces vs. natural language user interfaces;
- Supervision: low autonomy vs. high autonomy without supervision;
- System design and dynamic control-flow: no interaction with other tools and components vs. conversationally interaction with other agents.

LxM agents give a new boost to the idea of *robotic process automation* (RPA), as an RPA system is traditionally based on simple rule-based approaches (Czarnecki and Fettke 2021; Mendling et al. 2018).

Another often referenced concept in our context is the idea of so-called *foundation models* which describe pre-trained models that can be used for several so-called *down-stream tasks* such as question-answering, sentiment analysis, information extraction, image captioning, object recognition, or instruction following without the need for particular model adaptation or fine-tuning (Bommasani et al. 2022). These models are trained on mass data in an unsupervised way and nowadays often encompass several so-called modalities, e.g. text, image, speech, structured data, and signals. It is obvious that the capabilities of such a model provide an important component for agentic systems. However, similar to the fact that a human agent does not only consist of a human brain, but of more sensors and actuators, these sensors and actuators open the possibility to interact with the environment. Without it, the human agent would not be able to act in its environment. Similarly, an agentic system needs, beside such a foundation model, other components to really act as an agent.

The next section of this editorial will give a more technical background and details of several

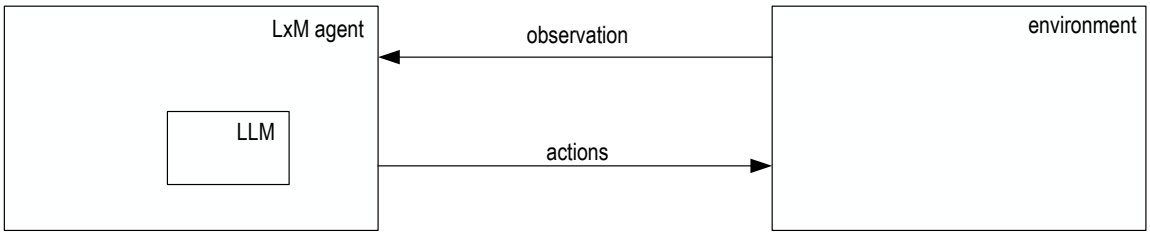

Figure 1: Abstract LxM agent architecture

LxM agents. Section 3 will discuss the importance of conceptual modeling in this context. Our editorial closes with some conclusions and important research questions.

2 Technical Background

In this section, we give more details on the technical background of an LxM agent. Therefore, we proceed in three steps. First, in Section 2.1 we introduce an abstract architecture. Second, in Section 2.2 we illustrate the abstract architecture by various examples for LLM-based agents of different complexities, which were implemented by the authors. Third, Section 2.3, describes some dedicated LxM Agents developed in the literature.

2.1 Abstract LxM Architecture

Fig. 1 shows the general description of an LxM agent. As already argued in the Introduction, an LxM agent consists of an LLM, can *observe* an environment, e.g., via sensors, and manipulate the environment via *actions*, e.g. through actuators. The specific technical details of these modules are very different from one particular system to another.

2.2 LLM-based Agents: Basic Agent examples

LxM agents make use of the fact that mainstream LLM are not only trained to act as chatbots, but also to follow instructions and call external services resp. tools, cf. (OpenAI 2025). However, the LLM infrastructure does not perform the calls to services itself. Instead, the LLM respond with the service it wants to call together with all the parameters required for an invocation. The agent's execution environment then performs the invocation and replies to the LLM with the return value of the tool call.

We first provide an example for the basic LxM agent architecture in Figure 2 and Figure 3. For illustrative purposes, this agent is created in a classical chat bot environment. The agent is based on an instruction prompt using a task description and a set of actions. The task description may include text as well as symbolic information, e.g., in the form of a BPMN diagram. It instructs the LLM to act like an agent. Instead of replying to a user request in textual form, the agent is only allowed to reply with one of the actions supplied with the prompt. Whenever the agent replies to a message, the execution environment will invoke the action/service and send a new message to the agent with the reply of the service invocation.

The Figures 4 and 5 show two possible user interactions with the agent defined in Fig. 3. In 4, the user asks the agent to start the heating thirty minutes before his last appointment ends. The bot autonomously decides to retrieve the user's schedule via an ICAL resource. After the resource is provided to the agent by the execution environment, the agent calls a generic scheduling activity where the first parameter is the turn-on heating activity and the second parameter is the derived time-point.

In the second example shown in Figure 5, the user asks the agent to remember his daughter to

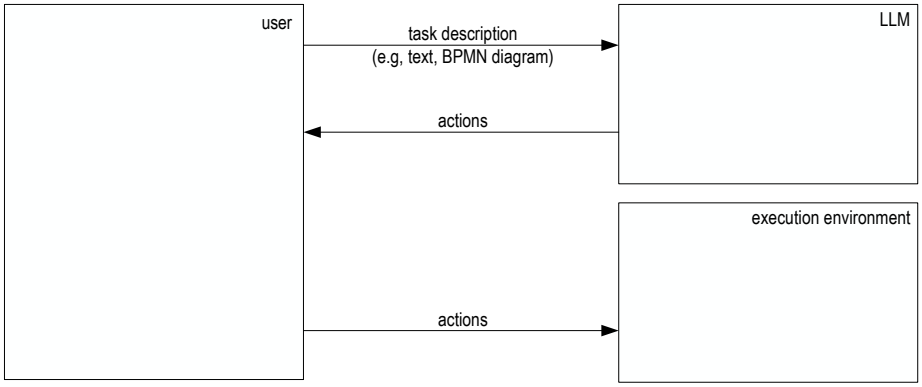

Figure 2: Architecture of a basic LLM-based agent

wear sunscreen if the weather is sunny at three o' clock on that day. The agent autonomously decides to schedule the calling of a vision agent (an agent based on a vision model) for 3PM. It passes the webcam image of the user´s home town as the first parameter and asks for if the weather is sunny. Once the scheduled call to the vision agent is completed, the environment sends a reply to the agent. In the example the weather is sunny, so the agent calls an action to send an email to the user's daughter.

2.3 Agent Example with Tool Calling

The previous example was meant for illustrative purposes only. It can be tried with virtually every LLM-based chatbot application. However, real agents are realized with the help of explicit tool definitions. This means the LLM API typically receives the available tools as a separate parameter and assumes that the parameters of the tools are defined in the form of JSON Schema. Using the OpenAI Completions API we can define the agent from the previous example with the message shown in Listing 1. However, the basic principle does not change.

The major difference is that the LLM answers with a specific message containing only the required tool call. The receiver can then execute the requested function locally and provide the result to the LLM.

Over time, a large number of agent frameworks were introduced. Such frameworks typically provide features such as multi-agent collaboration, an environment for tool calling, RAG workflows, and unified LLM interfaces. The Github Repository[1] provides an excellent overview.

```
{
  "model": "gpt-4o",
  "messages": [
    {
      "role": "system",
      "content": "You are a helpful agent. \n The
          following URLs may be accessed via the fetch
          () action: \n http://mycalendar.com/user.ics
           // the calendar of the current user in ICAL
          format \n http://mytwon.com/webcam.jpg    //
          the current webcam picture showing the
          current weather conditions. \n No other
          resource must be used. You are not a vision
          model. The time zone of the user is Europe/
          Berlin. \n For every message you receive,
          always reply only with one of the above
          actions. Make sure to only execute actions
          once all required information is available.
          You may first need to request external
          information via fetch() or by asking the user
           via ask(). You must not reply to the user
          without calling an action. Do not provide any
           other output. Reply with exactly one action
          ."
    },
    {
      "role": "user",
      "content": "Turn on my heating 30 minutes before
          my last appointment ends."
    }
  ],
  "tools": [
    {
      "type": "function",
      "function": {
        "name": "fetch",
        "description": "Access a web resource.",
```

[1] https://github.com/kaushikb11/awesome-llm-agents

System User:

You are a helpful agent. Only the following actions are available:
- fetch(URL,httpMethod) // acccess a web resource
- schedule(action(), timepoint) // schedule an action at an absolute timepoint DD:MM:YY H:M:S
- ask(query) // ask the user
- callVisionAgent(imageurl, yourMessage) // ask another agent about a picture
- turnHeatingOn // turn the heating system of the user only
- turnHeatingOff turn the heating system of the user only
- sendMessage(email, subject, body)

The following URLs may be accessed via the fetch() action:
- http://mycalendar.com/user.ics // the calendar of the current user in ICAL format
- http://mytwon.com/webcam.jpg // the current webcam picture showing the current weather conditions.

No other resource must be used. You are not a vision model. The time zone of the user is Europe/Berlin.
For every message you receive, always reply only with one of the above actions. Make sure to only execute actions once all required information is available. You may first need to request external information via fetch() or by asking the user via ask(). You must not reply to the user without calling an action. Do not provide any other output. Reply with exactly one action.

Figure 3: Instruction Prompt for a simplified Smart Home Agent

Agent:
ask("What would you like help with today?")

User:
Turn on my heating 30 minutes before my last appointment ends.

Agent to Execution Environment:
fetch(http://mycalendar.com/user.ics, "GET")

Execution Environment to Agent:
BEGIN:VCALENDAR
...
DTSTART;TZID=Europe/Berlin:20250304T160000
DTEND;TZID=Europe/Berlin:20250304T190000
SUMMARY:LxM Agent Project Meeting
END:VEVENT
END:VCALENDAR

Agent to Execution Environment:
schedule(turnHeatingOn, "04.03.25 18:30:00")

Figure 4: Example Interaction 1 with Agent from Fig. 3

```
        "parameters": {
          "type": "object",
          "properties": {
            "URL": {
              "type": "string"
            },
            "httpMethod": {
              "type": "string"
            }
          },
          "required": [
            "URL",
            "httpMethod"
          ],
          "additionalProperties": false
        },
        "strict": true
      }
    },
    ...
  ],
  "store": false
}
```

Listing 1: Example message to LLM using OpenAI API tool calling

2.3.1 Generic Agent Example

In the previous examples, we used a hand-crafted instruction prompt to define the capabilities and behavior of the agent. Such an instruction prompt may be derived from some kind of model. However, it is also possible to take a model as an input for a generic agent. An example of such an agent is shown in Figure 6. We have evaluated this prompt via ChatGPT (GPT 4o). The user provided the Process Model shown in Figure 8 as a *PNG* file. We were then able to execute the same behavior for the heating and sunscreen examples as in the previous section. It is worth noting that the agent was not able to follow BPMN XML correctly. It is, therefore, an interesting topic to explore how models should be communicated with agents.

2.3.2 Vision enabled IOT Agent creating a model

The next example pertains to the area of Smart Living and the support of tasks in households, e.g. via robots or other assistance systems (Fig. 9). For this example, the Ferret-13B LLM has been used, which is a multi-modal, foundational LLM that can understand and process spatial references

Agent:
ask("What would you like help with today?")

User:
My daughter will play outside today at 3 pm. Please check the weather when she starts playing, and if it is sunny, send her a reminder to wear sunscreen.

Agent to Execution Environment:
schedule(callVisionAgent('http://mytwon.com/webcam.jpg', 'is there sun on this image?'), 15:00:00 12-10-23)

Execution Environment to Agent:
yes

Agent to Execution Environment:
sendMessage("mydauthger@myfamily.com", "Reminder to Wear Sunscreen", "The weather is sunny! Don't forget to wear sunscreen while you're playing outside today. Enjoy your time!")

Figure 5: Example Interaction 2 with Agent from Fig. 3

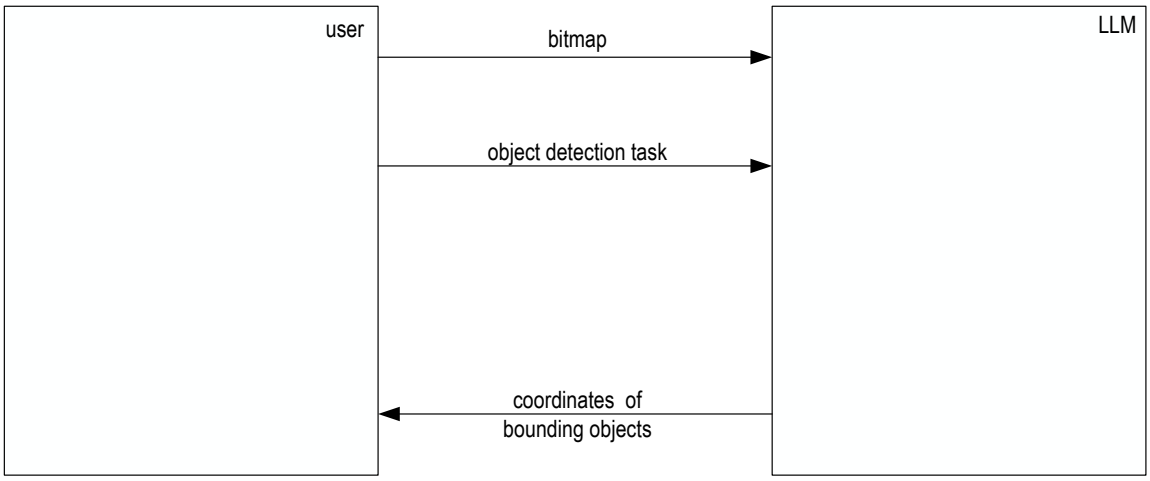

Figure 6: LLM-based vision architecture

System User:
You are a helpful agent called "AI Agent".
Your job is to behave as described in the BPMN diagram that will be provided next by the user. First ask for the model. Then interpret the model and behave according to the definition. If you need external data, respond with call <activitylabel(parameter)> to the user. The execution environment will then execute the activity and provide you the result as a user message. User interactions can only be performed via the activities of the User lane. Also, answer with call <activitylabel(parameter)>. The user will then provide the required information. Each Response from your side must only contain an allowed activity either of the user or of the execution environment lane.

Agent:
Please provide the BPMN model.

User:
Provides Diagram as PNG Image

Figure 7: Generic Agent interpreting a BPMN Diagram

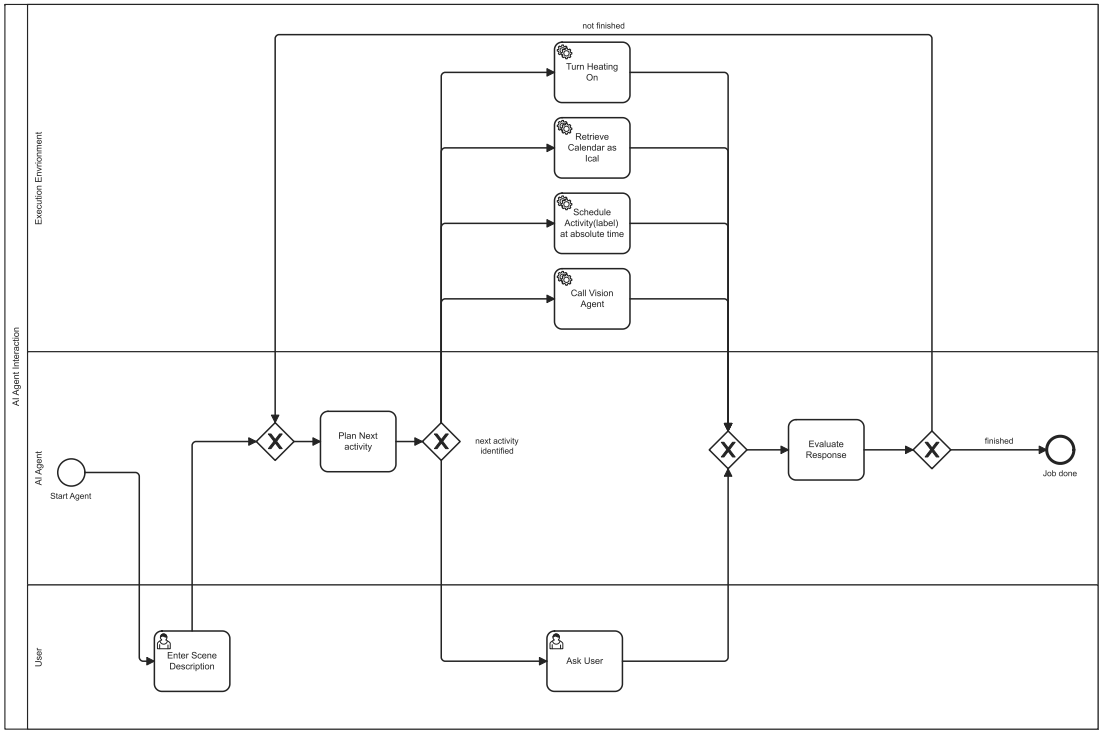

Figure 8: BPMN Model describing Agent Behavior as input for Generic Agent in Fig. 7

without having to be trained on specific objects or environments (You et al. 2023). It is provided by Apple for free for research purposes[2] and can be run on consumer hardware. As depicted in Figure 10, the input to the LLM was a picture of a bathroom in which the positions of different objects shall be identified. By issuing prompts asking for the coordinates of several objects, the LLM returns the position information in the form of coordinates for bounding boxes. These can serve as a basis for creating an according model as shown in the last image (7) of Figure 10. The LLM can also be used for identifying objects in certain areas, e.g. by handing it an image together with a region in that image. In contrast to previous approaches in computer vision, this LLM does not require particular pre-training for recognizing objects. By adding the model aspect in the agent pipeline, it can be verified which objects and regions an LLM has recognized and how this information serves as input for actions.

2.3.3 Vision-enabled Agent for Planning Actions

The example in Figure 11 shows how a vision-enabled LLM can be used to first derive the steps of a cooking process with references to objects in a given image. Subsequently, this information about the cooking process is transformed into a BPMN process model. This representation in the form of a conceptual model permits to more easily identify paths and potential errors in the process (Fill et al. 2023, 2024). We generated the BPMN model based on the agent's input using an extended version of (Köpke and Safan 2025).

2.4 Dedicated LxM agents

The following subsections provide an overview of the dedicated LxM agents found in the literature.

[2] https://github.com/apple/ml-ferret

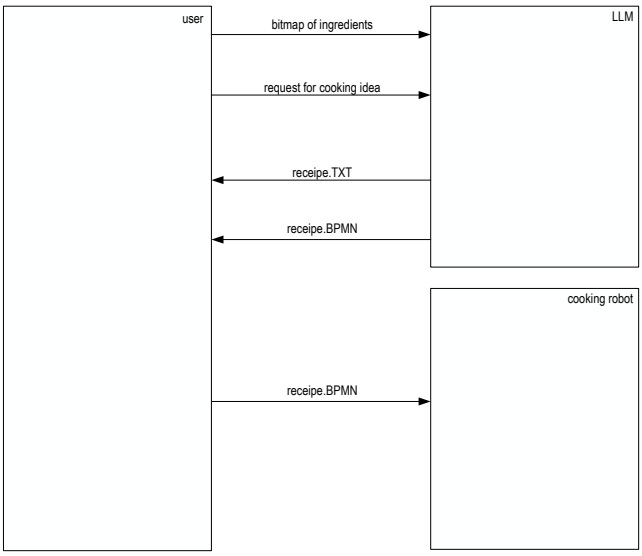

Figure 9: LLM-based vision architecture

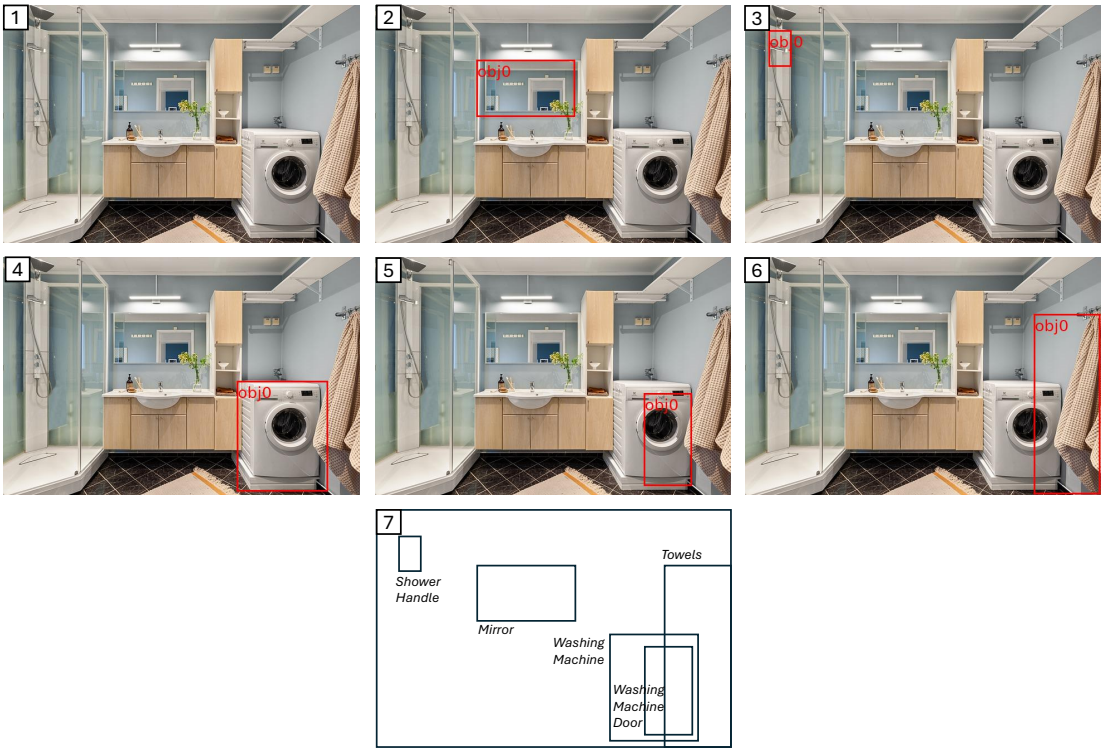

Figure 10: Example for model elicitation using the Ferret-13B LLM: Coordinates retrieved via prompts on the original image [1] of objects (mirror [2], shower handle [3], washing machine [4], washing machine door [5], and towels [6] were used as a basis for the model in [7]. Original image by Pixabay, Source: https://pixabay.com/photos/real-estate-interior-bath-room-6900973/

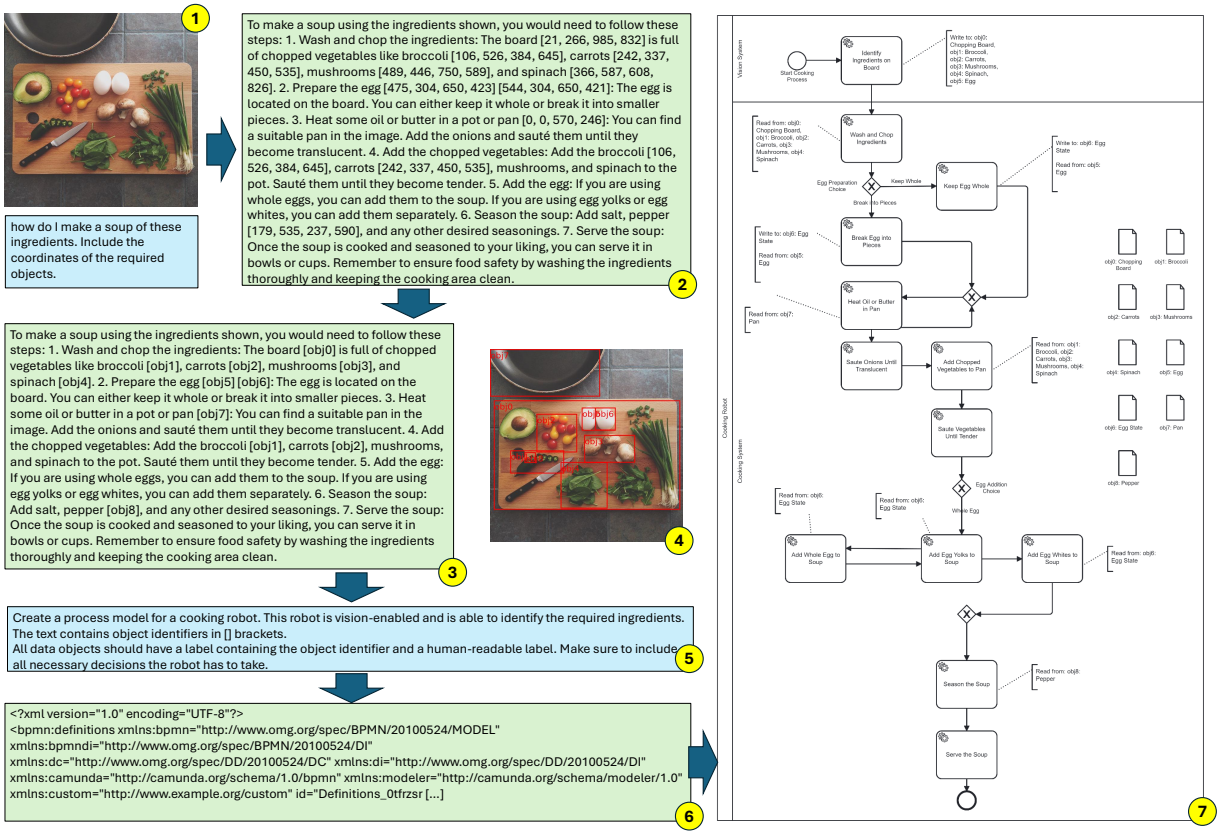

Figure 11: Example for the LLM-based derivation of a BPMN process model for cooking instructions including references to coordinates of objects using Ferret-13B and GPT-4o. Original image by Pixabay, Source: https://pixabay.com/photos/avocado-chopping-board-ingredients-1838785/

2.4.1 LxM agent WebShop

An example is shown in Fig. 12, an agent which is able to buy products in a web shop. The agent consists of an LLM, ResNet, and a web-query generation. Its sensors and actuators communicate with the environment by HTTP requests and responses. The agent is able to fulfill tasks such as "I want a small portable folding desk".

2.4.2 LxM agent UFO

Fig. 13 shows the system UFO developed by Microsoft (C. Zhang et al. 2024). This system additionally is composed of a planner which is able to plan future actions. Its environment is a windows desktop. As observational input, windows screens and bitmaps are used. UFO allows to manipulate the environment by executing particular mouse moves, clicks, or keyboard strokes. Typical tasks which can be executed by UFO are "close all open PDF documents", "delete all notes in my presentation", or "draft an email to John thanking him for the preprint on LxM agents."

2.4.3 LxM agent YuraScanner

A third, and last example, is depicted in Fig. 13. It shows an LxM agent which is autonomously able to scan unknown web applications and detect security vulnerabilities in a blackbox manner. Therefore, the agent interacts with the web application via HTTP requests and responses, autonomously identifies the possible navigation on the side, as well as possible workflows to fulfill particular tasks, and – finally – to detect security problems in these workflows. Note, that the more detailed description of the last system is due to the fact that more information is publicly available.

2.5 Some intermediate remarks

Summarzing our presented example cases and many more cases described in practice and academia, we have to admit that LLM in general and the recent from talking machines to acting machines powered by LLM is often confusing and it is difficult to understand how a system exactly works and how it performs under realistic circumstances. One reason for the opacity of the technology is that the particular components which are used, adapted, or enhanced are often not clear. From a technological point of view, this situation is of course surprising since the technical implementation needs to know which components builds on other components. However, having in mind the fact that the market for AI in general and LLM in particular, is estimated in billions of Euros, Pounds, US-Dollar etc., this is not surprising.

To shed some light on the different systems, Fig. 15 gives an overall overview of the systems described before. This figure relies on the idea of (Bommasani et al. 2023) to describe the ecosystem of foundational models by a so-called "ecosystem graph". Each node of such a graph represents an asset:

- Dataset asset: data used for training purposes,
- Model asset: a pre-trained machine learning model,
- Application asset: an applications built on top.

An arrow between two nodes indicates that one asset depends on the existence of another asset.

The overview clearly shows that GPT4 currently provides a central hub in developing the LxM agents presented. However, the provided WebShop example does not use it. One reason may be that this is an early example. Additionally, it is interesting to combine different LLM in one application, as our example shows. Nevertheless, it is also possible to use other LLM for building agents. Last but not least, in our scenario, no particular datasets for LxM agents are used. Note, that the ecosystem graph provides instances of such assets, e.g., human and robots trajectories, workflows (Bommasani et al. 2023).

Based on these remarks, it is pretty clear that in the future a better understanding of the LxM agents is necessary. Therefore, we discuss in the following the role of modeling for this purpose.

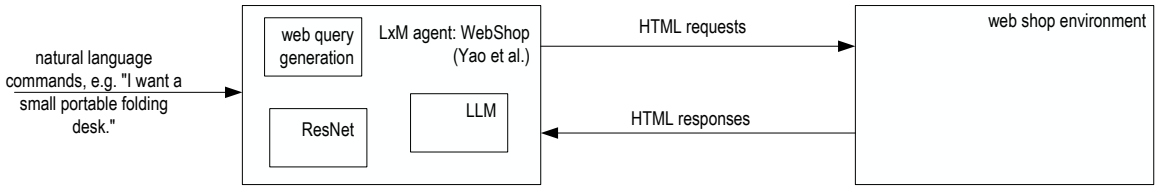

Figure 12: Architecture for the LxM agent WebShop (based on: Yao et al. 2023)

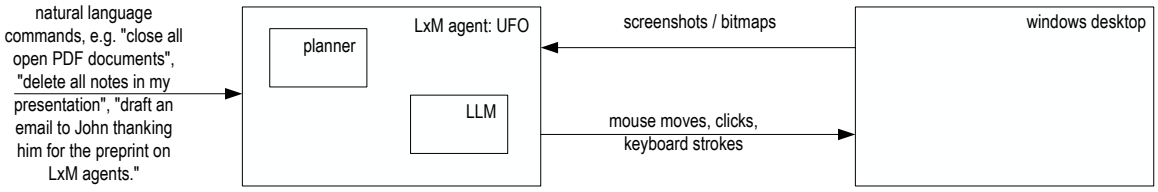

Figure 13: Architecture for the LxM agent UFO (based on: C. Zhang et al. 2024)

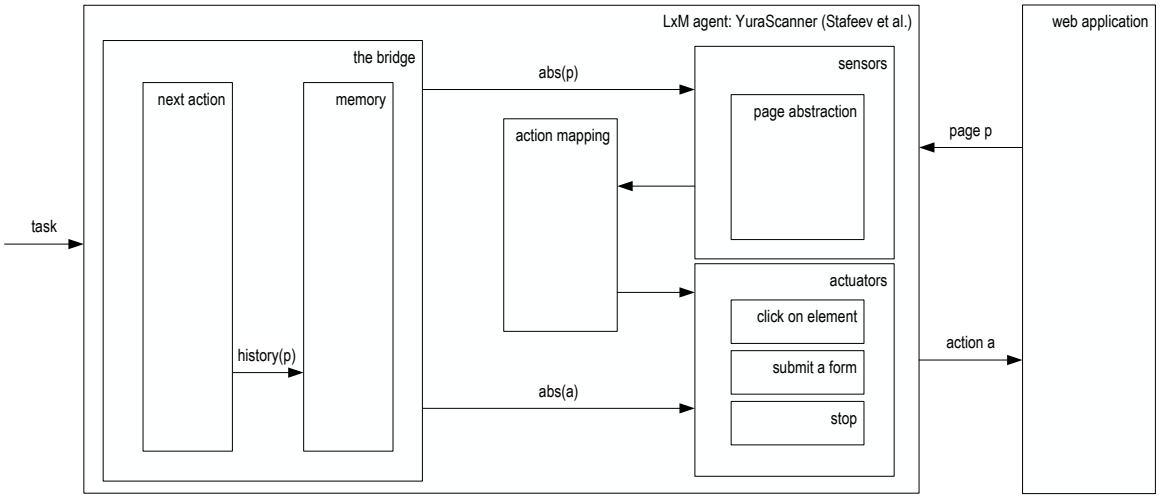

Figure 14: Architecture for the LxM agent YuraScanner (based on: Stafeev et al. 2025)

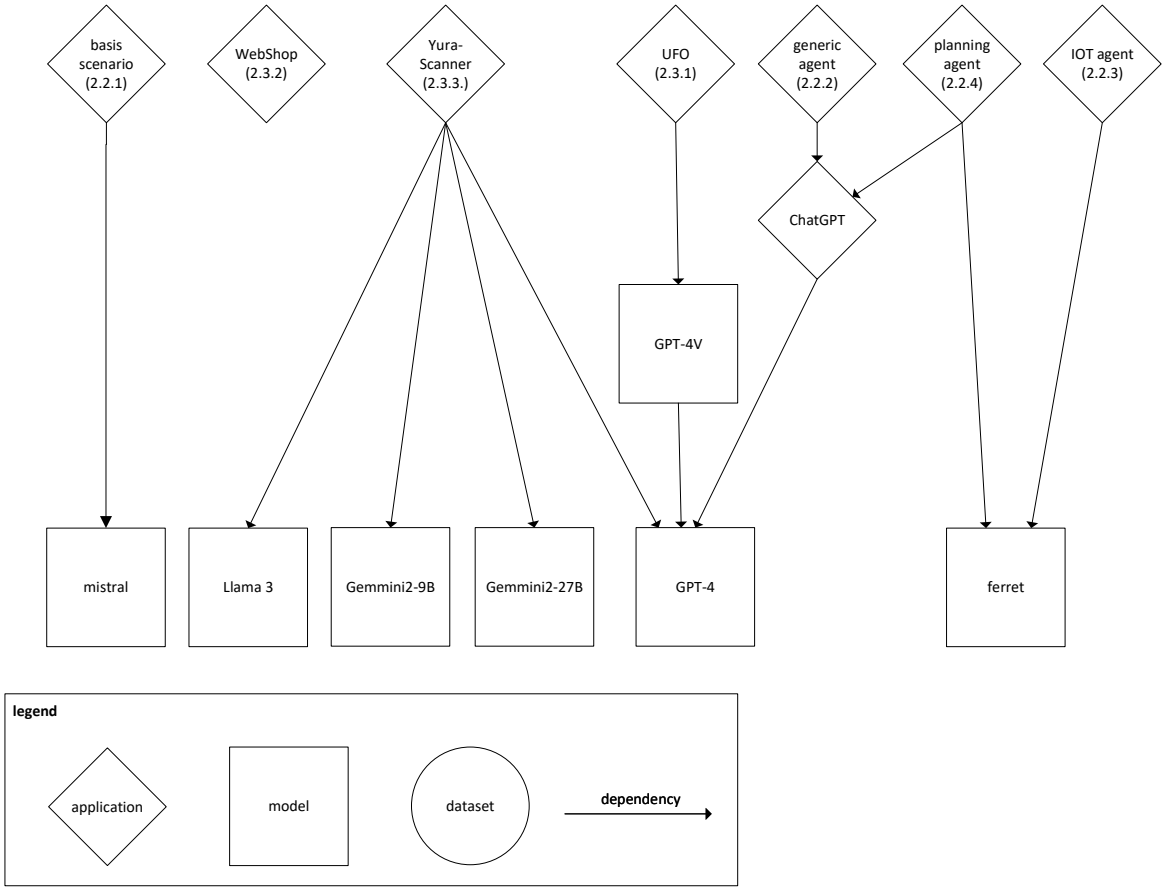

Figure 15: Ecosystem graph for our examples described in this Section, extends the idea of Bommasani et al. 2023

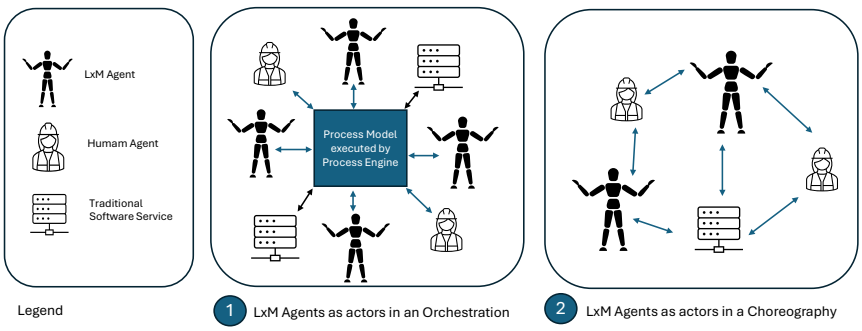

Figure 16: LxM agents as actors of choreographies or orchestrations

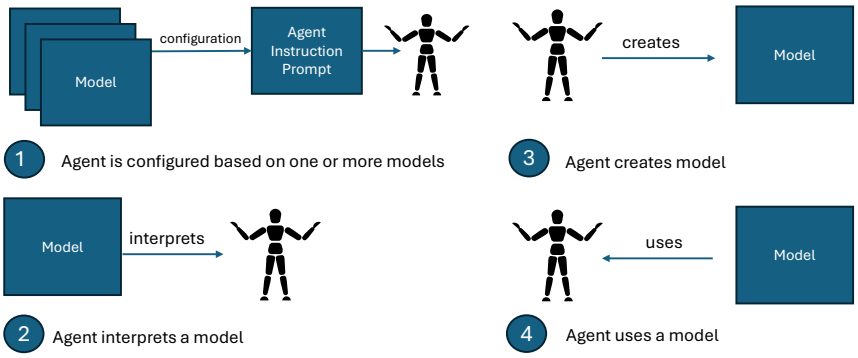

Figure 17: Modeling meets LxM Agents

3 LxM Agents and Modeling

In the following, we discuss how conceptual modeling can contribute to the landscape of LxM agents.

3.1 LxM agents as actors of processes

LxM agents can be actors in orchestrations (Fig. 16(1)) or choreographies (Fig. 16(2)). We call processes partially based on LxM agents 'LxM agent extended processes'. An LxM agent may execute a specific task of an orchestration. However, the triggering of the task execution lies in the control of the orchestration engine. In this sense, the LxM agent is an alternative implementation of an RPA agent. Furthermore, an LxM agent may execute several tasks within an orchestration, just like a human actor. The capability of an LxM agent to remember previous actions shows a significant difference between an automated task and an LxM agent. It may, therefore, be helpful to model LxM agents in their own lanes in a BPMN diagram. However, an even more dynamic approach - i.e., more agentic - is to model LxM agents as actors in a choreography. In this case, they can freely decide when and what to execute if their public behavior adheres to the choreography specification. It should be noted that both patterns are not necessarily disjoint: One agent may simultaneously participate in multiple orchestrations or choreographies.

We, therefore, argue that modeling is essential to guarantee the correct execution of LxM agent extended processes. Therefore, the first area where conceptual modeling can contribute is to explore how to model various kinds of LxM extended processes. This does not only relate to the global view of orchestrations or choreographies but also the modeling of agents themselves. We have identified four scenarios where agents interact with models - see Figure 17.

3.2 LxM agents configured with models

LxM agents are typically programmed by manually providing textual instruction prompts. An alternative solution is the automatic configuration of agent instructions based on models. The input models may for example be process models of various types. This is especially relevant for guaranteeing that the agent's behavior complies with a process model in the form of a choreography or orchestration. However, the models are not restricted to process models. Also, data models are highly relevant for supporting actionable agents that can automatically enact services. Currently, LLM agents typically support the definition of message formats, e.g., using JSON schemas.

3.3 LxM agents interpret models

An alternative to Subsection 3.1, where instruction prompts are configured from models, is the usage of generic agent prompts, where the agent's behavior is provided as input in the form of a model at runtime (Fill et al. 2024). This differs from the previous case, where an agent is configured based on a model due to two reasons: (1) The model is

provided as an input at runtime and not via the instruction prompt. (2) The agent interprets the model itself and can thus completely change its behavior based on the provided input model.
Subsection 2.3.1 provides an example of a generic agent prompt that, as its first interaction with the user, asks for a process model. Once the process model is uploaded, the agent behaves as defined in that provided model.

3.4 LxM agents create models

Current LxM agents may query data from the environment and execute services. It is only a minor step when, instead of simple services, the agents generate models (Fill et al. 2023, 2024). One example is the generation of a process model for the execution of a sub process. This approach has multiple benefits: 1) classical model checking methods can be applied on the generated models (safeness, soundness, etc.). Furthermore, the computationally expensive agents are not bothered with the execution of processes. Finally, the generated models can be communicated with human users to support explainable AI. The ability of LLM to generate high quality process models was already shown in approaches such as (Köpke and Safan 2025; Kourani et al. 2024).

3.5 LxM agents process models

Finally, any kind of model can be used by an LxM agent: E.g., an agent that should translate messages following heterogeneous schemas may request the corresponding reference models to find a mapping on the level of the models. It should be noted that "Agent interprets model" (Case 2 in Fig. 17) is a subcase of "Agent uses model" (Case 4). However, Case 2 is stronger as it implies that an input model defines the agent's behavior. In Case 4, an agent uses models for specific sub-tasks.

4 Conclusions

The examples shown before demonstrate two things: First, the integration of LLM in agent systems seems to be a fruitful combination. Although many scenarios still seem to be a kind of science fiction, particular evaluations of some concrete tasks demonstrate not only that such systems can be built in principle but also that the evaluation results show a significant performance in some tasks.
Secondly, our work showcases that conceptual modeling can significantly contribute to this vividly developing field of research.
In the future, there will be many challenges for research on the topic of LxM agents. These include, for example:

- Concepts for enhancing the degree of autonomy of LxM agents,
- Developing of techniques for guardrails for LxM agents,
- Quality assurance, checks and approvals of an LxM agent,
- Questions of trustworthiness and security of an LxM agent,
- Training of a hybrid LxM agent,
- Evaluations of LxM agents in laboratory and field environments,
- co-operation between different LxM agents.

Besides these technical questions, it is obvious that conceptual modeling can play a major role in these research streams. Models provide the instruments for understanding the world of LxM agents.

References

Bommasani R., Hudson D. A., Adeli E., Altman R., Arora S., Arx S. v., Bernstein M. S., Bohg J., Bosselut A., Brunskill E., Brynjolfsson E., Buch S., Card D., Castellon R., Chatterji N., Chen A., Creel K., Davis J. Q., Demszky D., Donahue C., Doumbouya M., Durmus E., Ermon S., Etchemendy J., Ethayarajh K., Fei-Fei L., Finn C., Gale T., Gillespie L., Goel K., Goodman N., Grossman S., Guha N., Hashimoto T., Henderson P., Hewitt J., Ho D. E., Hong J., Hsu K., Huang J., Icard T., Jain S., Jurafsky D., Kalluri P., Karamcheti S., Keeling G., Khani F., Khattab O., Koh P. W., Krass M., Krishna R., Kuditipudi R., Kumar A., Ladhak F., Lee M., Lee T., Leskovec J., Levent

I., Li X. L., Li X., Ma T., Malik A., Manning C. D., Mirchandani S., Mitchell E., Munyikwa Z., Nair S., Narayan A., Narayanan D., Newman B., Nie A., Niebles J. C., Nilforoshan H., Nyarko J., Ogut G., Orr L., Papadimitriou I., Park J. S., Piech C., Portelance E., Potts C., Raghunathan A., Reich R., Ren H., Rong F., Roohani Y., Ruiz C., Ryan J., Ré C., Sadigh D., Sagawa S., Santhanam K., Shih A., Srinivasan K., Tamkin A., Taori R., Thomas A. W., Tramèr F., Wang R. E., Wang W., Wu B., Wu J., Wu Y., Xie S. M., Yasunaga M., You J., Zaharia M., Zhang M., Zhang T., Zhang X., Zhang Y., Zheng L., Zhou K., Liang P. (July 2022) On the Opportunities and Risks of Foundation Models. http://arxiv.org/abs/2108.07258

Bommasani R., Soylu D., Liao T. I., Creel K. A., Liang P. (2023) Ecosystem Graphs: The Social Footprint of Foundation Models. https://arxiv.org/abs/2303.15772

Brown T. B., Mann B., Ryder N., Subbiah M., Kaplan J., Dhariwal P., Neelakantan A., Shyam P., Sastry G., Askell A., Agarwal S., Herbert-Voss A., Krueger G., Henighan T., Child R., Ramesh A., Ziegler D. M., Wu J., Winter C., Hesse C., Chen M., Sigler E., Litwin M., Gray S., Chess B., Clark J., Berner C., McCandlish S., Radford A., Sutskever I., Amodei D. (July 2020) Language Models are Few-Shot Learners. In: (arXiv:2005.14165) arXiv:2005.14165 [cs] http://arxiv.org/abs/2005.14165

Czarnecki C., Fettke P. (eds.) Robotic Process Automation: Management, Technology, Applications. De Gruyter Oldenbourg

Ding Y., Zhang X., Amiri S., Cao N., Yang H., Kaminski A., Esselink C., Zhang S. (Dec. 2023) Integrating action knowledge and LLMs for task planning and situation handling in open worlds en. In: Autonomous Robots 47(8), pp. 981–997

Fettke P. (2024) Large process models. https://www.youtube.com/watch?v=aj41zcbsawA. Last Access: 2025-05-15, 18th Symposium and Summer School On Service-Oriented Computing (SommerSOC), 2024, Crete

Fettke P., Strohmeier S. (2022) HR robotic process automation In: Handbook of Research on Artificial Intelligence in Human Resource Management Strohmeier S. (ed.) Edward Elgar, pp. 187–206

Fill H.-G., Fettke P., Köpke J. (2023) Conceptual Modeling and Large Language Models: Impressions From First Experiments With ChatGPT. In: Enterp. Model. Inf. Syst. Archit. Int. J. Concept. Model. 18, p. 3 https://doi.org/10.18417/emisa.18.3

Fill H.-G., Härer F., Vasic I., Borcard D., Reitemeyer B., Muff F., Curty S., Bühlmann M. (2024) CMAG: A Framework for Conceptual Model Augmented Generative Artificial Intelligence. In: Gallinucci E., Yasar H., Liaskos S., Marcel P., Chen P. P., Cesare S. d., Gailly F. (eds.) Companion Proceedings of the 43rd International Conference on Conceptual Modeling: ER Forum, Special Topics, Posters and Demos Co-located with ER 2024, Pittsburgh, Pennsylvania, USA, October 28-31, 2024. CEUR Workshop Proceedings Vol. 3849. CEUR-WS.org, pp. 56–69 https://ceur-ws.org/Vol-3849/forum5.pdf

Jurafsky D., Martin J. H. (2025) Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition eng, Third Edition Draft. Draft https://web.stanford.edu/~jurafsky/slp3/ed3book_Jan25.pdf

Kampik T., Warmuth C., Rebmann A., Agam R., Egger L. N. P., Gerber A., Hoffart J., Kolk J., Herzig P., Decker G., van der Aa H., Polyvyanyy A., Rinderle-Ma S., Weber I., Weidlich M. (2023) Large Process Models: Business Process Management in the Age of Generative AI. In: CoRR abs/2309.00900 https://doi.org/10.48550/arXiv.2309.00900

Kapoor S., Stroebl B., Siegel Z. S., Nadgir N., Narayanan A. (2024) AI Agents That Matter. In: arXiv preprint arXiv:2401.00001

Köpke J., Safan A. (2025) Efficient LLM-Based Conversational Process Modeling. In: Gdowska K., Gómez-López M. T., Rehse J.-R. (eds.) Business Process Management Workshops. Springer Nature Switzerland, Cham, pp. 259–270

Kourani H., Berti A., Schuster D., van der Aalst W. M. P. (2024) Process Modeling with Large Language Models. In: Enterprise, Business-Process and Information Systems Modeling. Springer Nature Switzerland, Cham, pp. 229–244

Mendling J., Decker G., Hull R., Reijers H. A., Weber I. (2018) How do Machine Learning, Robotic Process Automation, and Blockchains Affect the Human Factor in Business Process Management? In: Commun. Assoc. Inf. Syst. 43, p. 19 https://aisel.aisnet.org/cais/vol43/iss1/19

OpenAI (2025) Function calling - OpenAI API en. https://platform.openai.com

Radford A., Wu J., Child R., Luan D., Amodei D., Sutskever I. (2019) Language Models are Unsupervised Multitask Learners. In: https://www.semanticscholar.org/paper/Language-Models-are-Unsupervised-Multitask-Learners-Radford-Wu/9405cc0d6169988371b2755e573cc28650d14dfe

Senaratna N. I. (2025) The Relatives of LLMs – Why the LXM Family Is Just Getting Started. https://medium.com/on-technology/the-relatives-of-llms-8556e46a663e

Shannon C. E. (1948) A Mathematical Theory of Communication. In: The Bell System Technical Journal 27(July), pp. 379–423

Smolensky P. (1990) On the Proper Treatment of Connectionism en In: Philosophy, Mind, and Cognitive Inquiry: Resources for Understanding Mental Processes Cole D. J., Fetzer J. H., Rankin T. L. (eds.) Springer Netherlands, Dordrecht, pp. 145–206 https://doi.org/10.1007/978-94-009-1882-5_6

Stafeev A., Recktenwald T., Stefano G. D., Khodayari S., Pellegrino G. (2025) YuraScanner: Leveraging LLMs for Task-driven Web App Scanning. In: 32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025. The Internet Society https://www.ndss-symposium.org/ndss-paper/yurascanner-leveraging-llms-for-task-driven-web-app-scanning/

Wahlster W. (2024) Hybrid LxM Technologies: AI as a Booster for Industrie 4.0. https://www.youtube.com/live/91-CZX__S-U. Last Access: 2025-05-15, Keynote, International Advisory Board Meeting of CIIRC/RICAIP, Czech Institute of Informatics, Robotics and Cybernetics (CIIRC CTU)

Wang L., Yang F., Zhang C., Lu J., Qian J., He S., Zhao P., Qiao B., Huang R., Qin S., Su Q., Ye J., Zhang Y., Lou J., Lin Q., Rajmohan S., Zhang D., Zhang Q. (2024) Large Action Models: From Inception to Implementation. In: CoRR abs/2412.10047 https://doi.org/10.48550/arXiv.2412.10047

Weaver W. (1949) The Mathematics of Communication. In: Scientific American 181(1), pp. 11–15

Yao S., Chen H., Yang J., Narasimhan K. (2023) WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents. https://arxiv.org/abs/2207.01206

Yenduri G., Ramalingam M., Selvi G. C., Supriya Y., Srivastava G., Maddikunta P. K. R., Raj G. D., Jhaveri R. H., Prabadevi B., Wang W., Vasilakos A. V., Gadekallu T. R. (2024) GPT (Generative Pre-Trained Transformer)— A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions. In: IEEE Access 12, pp. 54608–54649

You H., Zhang H., Gan Z., Du X., Zhang B., Wang Z., Cao L., Chang S.-F., Yang Y. (2023) Ferret: Refer and Ground Anything Anywhere at Any Granularity. In: arXiv preprint arXiv:2310.07704

Zhang C., Li L., He S., Zhang X., Qiao B., Qin S., Ma M., Kang Y., Lin Q., Rajmohan S., Zhang D., Zhang Q. (2024) UFO: A UI-Focused Agent for Windows OS Interaction. https://arxiv.org/abs/2402.07939

Zhang J., Lan T., Zhu M., Liu Z., Hoang T., Kokane S., Yao W., Tan J., Prabhakar A., Chen H., Liu Z., Feng Y., Awalgaonkar T. M., Murthy R., Hu E., Chen Z., Xu R., Niebles J. C., Heinecke S., Wang H., Savarese S., Xiong C. (2024) xLAM: A Family of Large Action Models to Empower AI Agent Systems. In: CoRR abs/2409.03215 https://doi.org/10.48550/arXiv.2409.03215

Zhao W. X., Zhou K., Li J., Tang T., Wang X., Hou Y., Min Y., Zhang B., Zhang J., Dong Z., Du Y., Yang C., Chen Y., Chen Z., Jiang J., Ren R., Li Y., Tang X., Liu Z., Liu P., Nie J.-Y., Wen J.-R. (Mar. 2025) A Survey of Large Language Models. In: (arXiv:2303.18223) arXiv:2303.18223 [cs] http://arxiv.org/abs/2303.18223