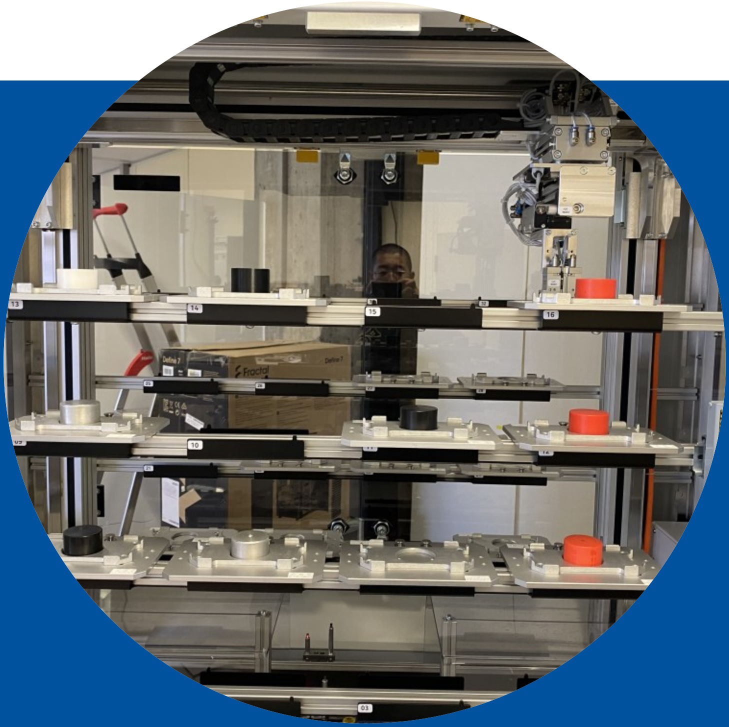
Universität Stuttgart
Institute of Industrial Automation
and Software Engineering

LLM-Powered Automation of Robotic Tasks in Warehouse



Zhe CAO
Elektromobilität
Supervisor: Yuchen Xia M.Sc.

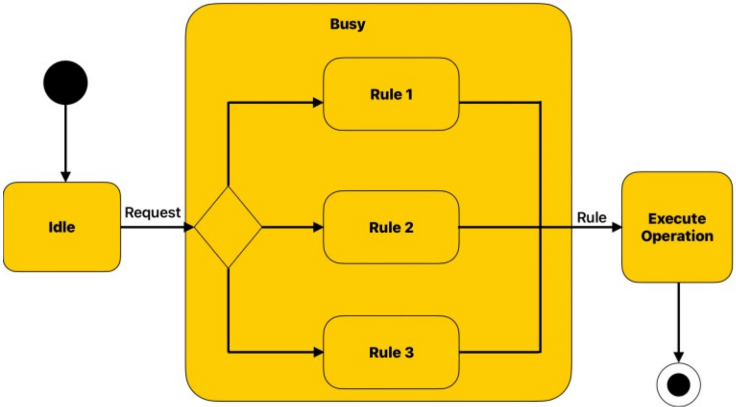
1. Background
2. Basis
3. Conceptual Design
4. Implementation
5. Evaluation and Discussion
6. Outlook and Summary

Background

The Evolution of Warehouse Management

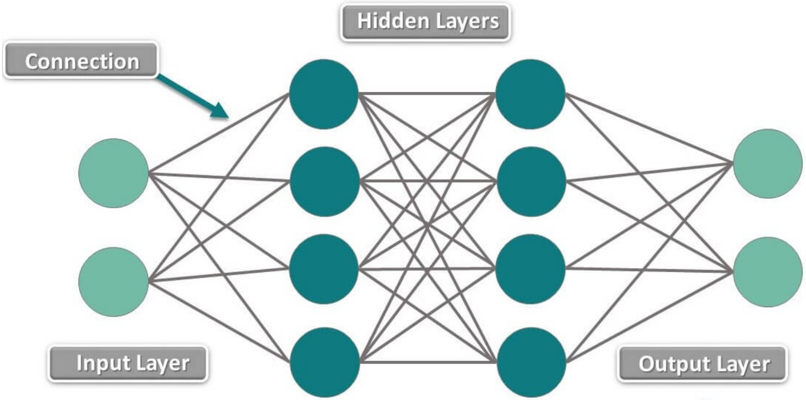
Traditional Warehouse

Rule-based sorting mechanism



High-level Automated Warehouse

Adaptive sorting system



[1]

LLM-Powered Automation

Semantic Interpretation + Task solving

1. Background
2. Basis
3. Conceptual Design
4. Implementation
5. Evaluation and Discussion
6. Outlook and Summary

What is LLM ?

Large language model is a type of artificial intelligence trained on a vast amount of text data, capable of understanding and generating natural language.

How can LLM help us ?

- Semantic Interpretation

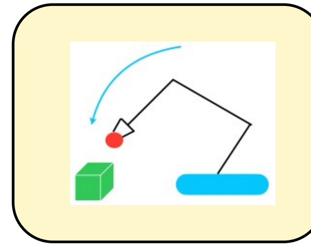
"input text" : "Please store these two golden watches for me"

text


```
{
  "task": "Store two golden watches",
  "data_query": true,
  "required_data": {
    "items": [
      {
        "position": "shelf",
        "size": "small",
        "color": "golden",
        "priority": "high",
        "value": "high",
        "condition": "normal"
      }
    ]
  },
  "task_type": "inbound"
}
```

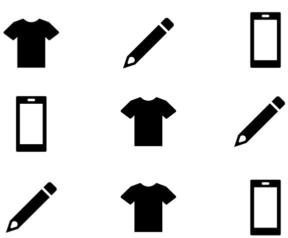

demand

Automation system



- Task solving

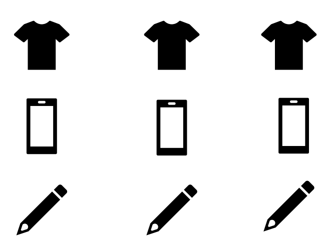
Sort items



planning

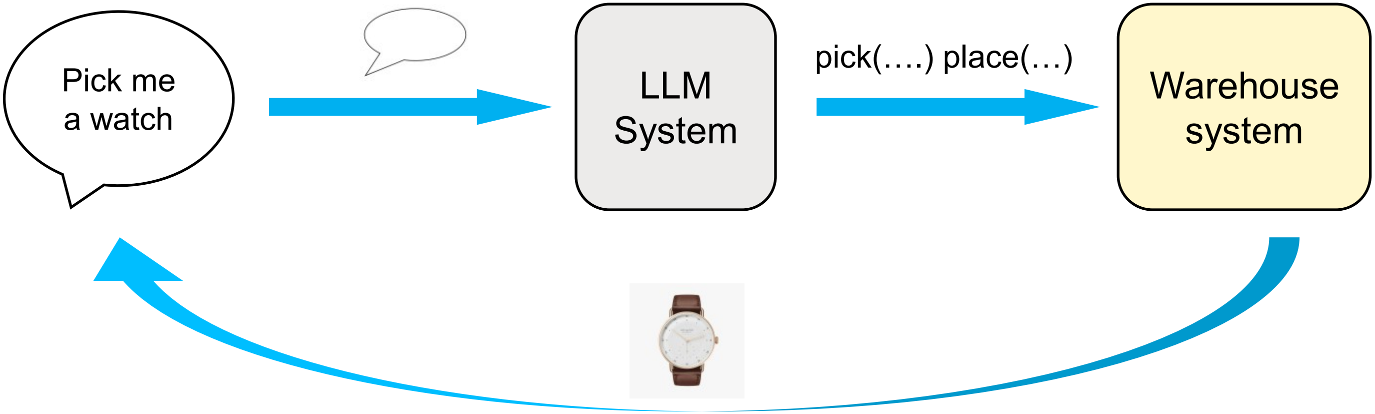
Control sequence

Result



1. Background
2. Basis
3. Conceptual Design
4. Implementation
5. Evaluation and Discussion
6. Outlook and Summary

System function

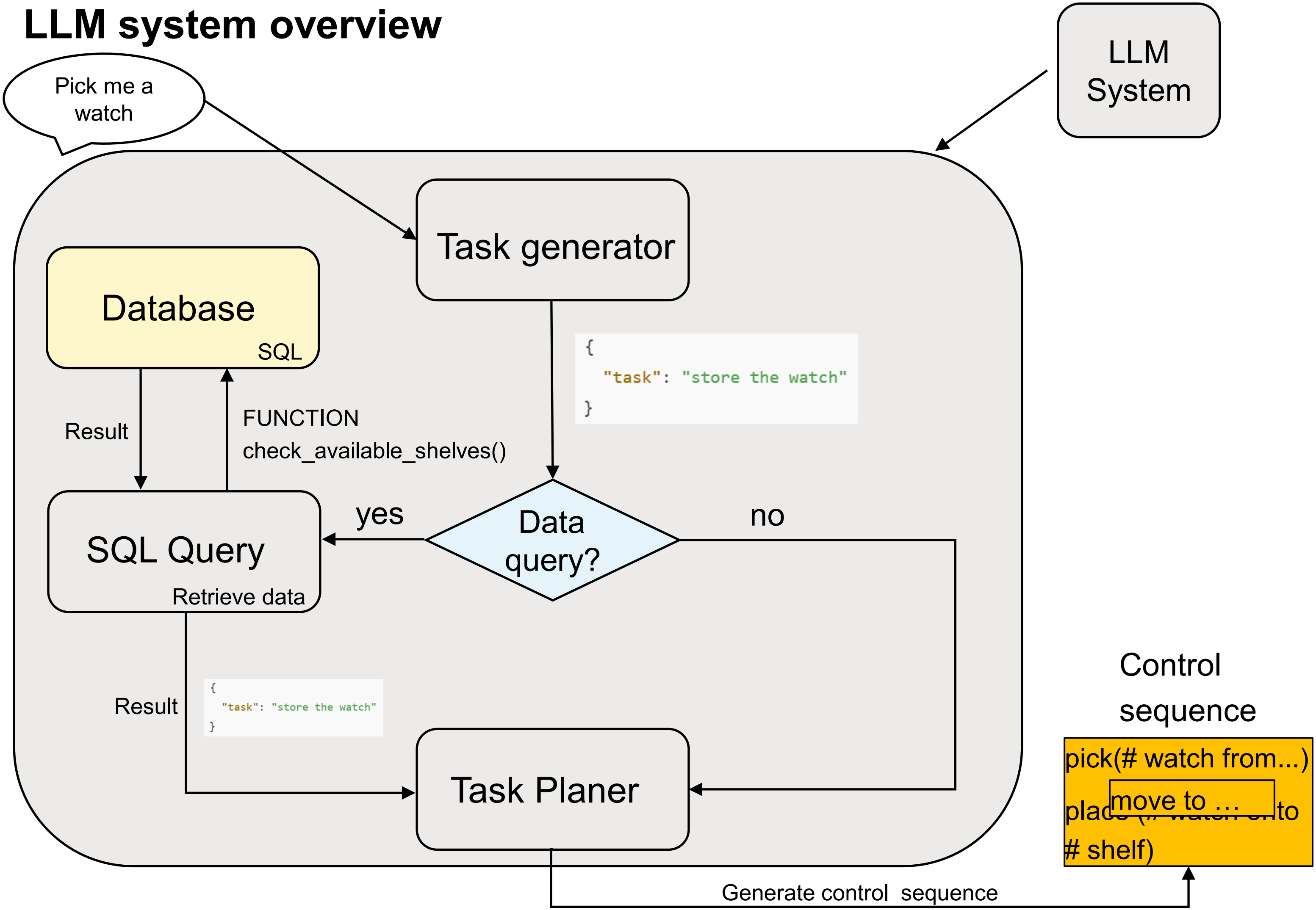


LLM system

Input: User request in natural language

Output: Control sequence

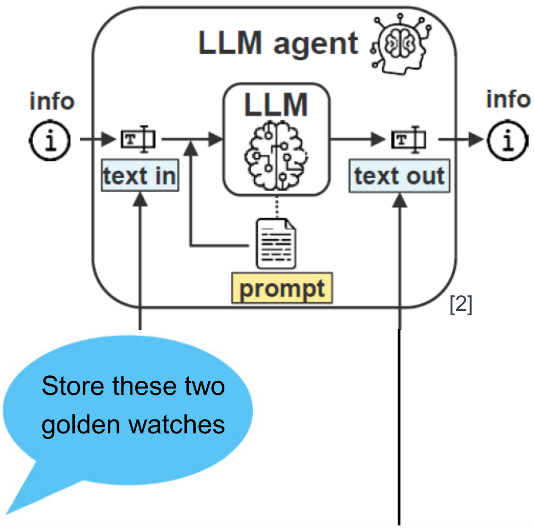
LLM system overview



How to build the LLM Multi-Agent system ?

Multi-agent system: **Task Generator** + SQL Query + Task Planer

1. Task Generator



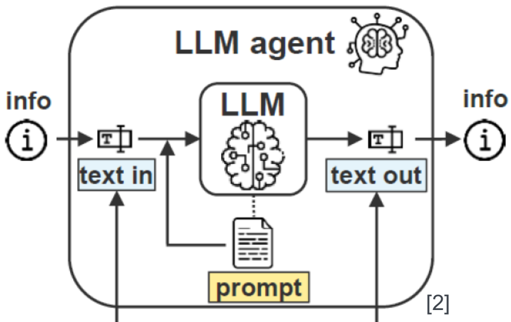

```json
{
  "task": "Find available shelves to store two expensive golden watches",
  "data query": "true",
  "required data": "{\"item_id\": \"golden_watch\", \"value\": \"high\",
  "task type": "inbound"
}
```

Role Definition	You are a task generator for a warehouse management system, creating tasks based on user requests.
Warehouse System Overview	**1**. Shelf Attributes: <shelf_id>, <realtime_temperature>, <realtime_humidity>, <realtime_light_intensity>; **2.** Item Attributes: <item_id>, <position>, <size>, <color>, <priority>, <value>, <condition>, <required-temperature>, <required-humidity>, <required-light-intensity>; **3.** All attribute data is stored in a SQL database, where it can be efficiently stored, queried, and managed.
Input	The essence of diverse user requests includes the following aspects: **1.** Organizing items based on specific requirements (…) **2.** Inbound and outbound of items (…) **3**. Emergency handling (…) **4.** Reset robotic arm (…) **5.** Degree of freedom check (…) **6.** Movement (…) **7.** Freeze (…) The first to third requests involve data query, while the fourth to seventh do not involve.
Output	The output requirements vary depending on the type of input: **1.** Inbound: First find out if there are shelves available for items to be stored, when yes, list all possible shelves; when not, return a denial of the request and explain the reason. (…)
Auxiliary Instruction	**1.** Return a JSON structure like: { "task": "...", "data query": "true/false", "required data": "{...some attributes of items...}" }; **2.** For the value of key "task", you should generate it based on the input-/output requirements above.(…)

How to build the LLM Multi-Agent system ?

Multi-agent system: Task Generator + **SQL Query** + Task Planer

2. SQL Query



[2]

```
{
  "task": "Find available shelves to store two expensive golden watches",
  "data query": "true",
  "required data": "{\"item_id\": \"golden_watch\", \"value\": \"high\",
  "task type": "inbound"
}
```

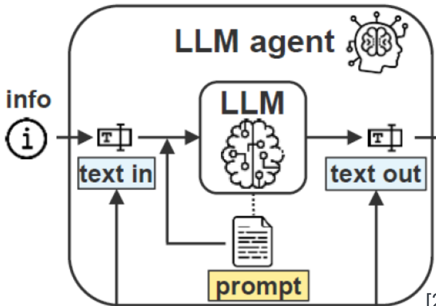
```
USE warehouse;

SELECT shelf.shelf_id
FROM shelf
LEFT JOIN item ON shelf.shelf_id = item.shelf_id
WHERE item.shelf_id IS NULL;
```

Role Definition	Convert JSON-defined data requirements into accurate SQL queries.
Warehouse System Detail	**1.** Tables and Attributes (…) item(id, size, color, value, priority, position, environmental condition) shelf(id, environmental condition) environmental condition(temperature, humidity, light intensity) **2.** Primary Key and Foreign Key Relationships (…) temperature→item; humidity→item; light intensity→item; shelf→item.
Input	A JSON structure including information about <task>, <task type>, and <required data>...
Output	For JSON Input: **1.** "task type":"inbound", first check if there are shelves available (means there's no item in the shelf and its condition fits for the storage) for items to be stored, you need to filter all shelves that meet the storage conditions based on the attributes of the items to be stored and output these shelves in a table format, If no shelves meet the requirements, your code should also indicate this. **2.** "task type":"outbound" (…) # Return a single explanatory message.
Auxiliary Instruction	**1.** Standard SQL Only: Ensure that the SQL query uses standard SQL constructs (e.g., CASE, EXISTS, UNION). Avoid procedural constructs like IF, THEN, or loops unless explicitly stated for stored procedures. **2.** Specific Conditions: Define all conditions explicitly in the query. Avoid placeholders like conditions_for_item_A and ensure all filtering logic is directly embedded. **3.** Always start your code with "USE warehouse" (…)

How to build the LLM Multi-Agent system ?

Multi-agent system: Task Generator + SQL Query + **Task Planer**

3. Task Planer



[2]

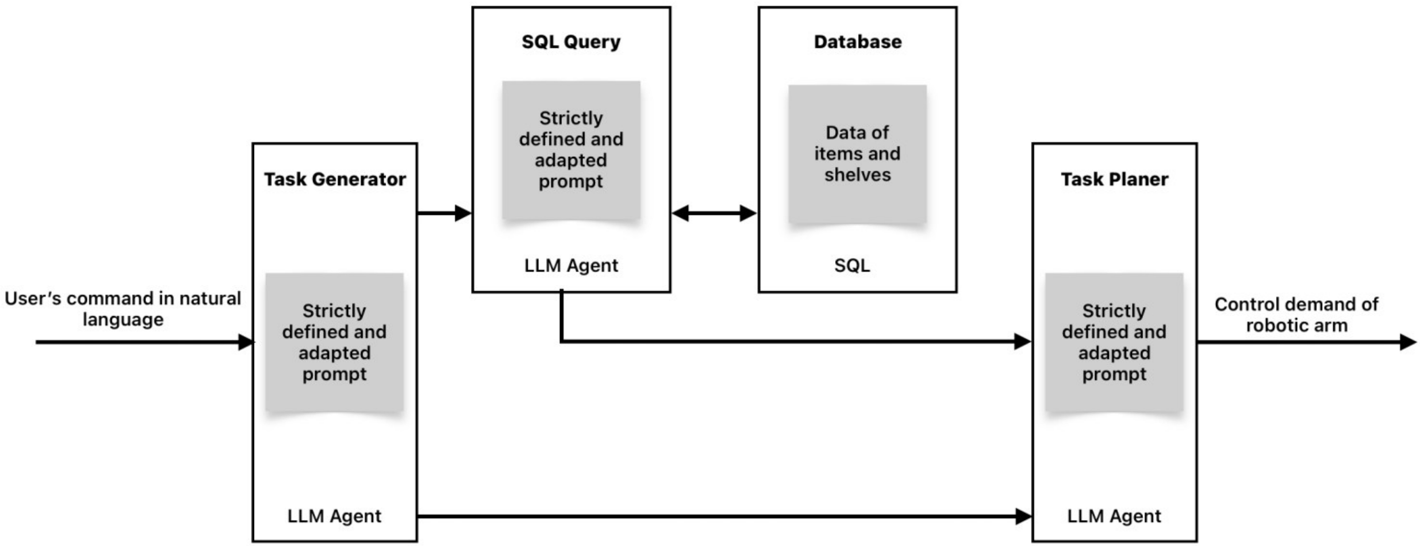
Result from database

```
"step 1": [
  "pick(golden_watch_1 from other place)",
  "place(golden_watch_1 to shelf2)"
],
"step 2": [
  "pick(golden_watch_2 from other place)",
  "place(golden_watch_2 to shelf4)"
],
```

Output	**1.**The function-sequence must be in a JSON structure like {"step1":["pick(#item_id from # item_location)","place(#item_id to #item_location)"]; "step 2":["pick(#item_id from #item_location)", "place(#item_id to #item_location)"] ...}. #item_location including two possibilities： <shelf>(in this case you also need to show <shelf_id> in the function-sequence) and <other place>. **2.** If an item cannot be placed on any shelf, include a "No suitable shelves currently available for item <item_id>." message in the output. **3.** Output the final distribution of items, with the position "other" for items that could not be placed on any shelf in the following JSON format: {"item-id": ..., "position": "shelf_id"; ...}.
Auxiliary Information	**1.** For standard operation procedure of organizing items based on specific requirements, make sure that all items, regardless of its position before(shelf/other place), are placed passing to the criteria given in different shelves sequentially. **2.** Clearly specify #item_id and #item_location at each pick-and-place function-call sequence. **3.** One shelf can only hold one item. **4.** Output the function-sequence in json form without explanation.

	"true", which means the task involves data in database. **2.** In the input in JSON format, the value of the key "data query" is "false", which means the task doesn't involve data in database.
Standard operation procedure	**1.** Inbound: 1) You get the information about the item to be stored and current available shelves from input. 2) You generate a function-call sequencen including pick and place, so that the item can be stored from other place to these shelves (use the name of the item(candy/cellphone/toy...) as its id) **2.** Outbound (…)

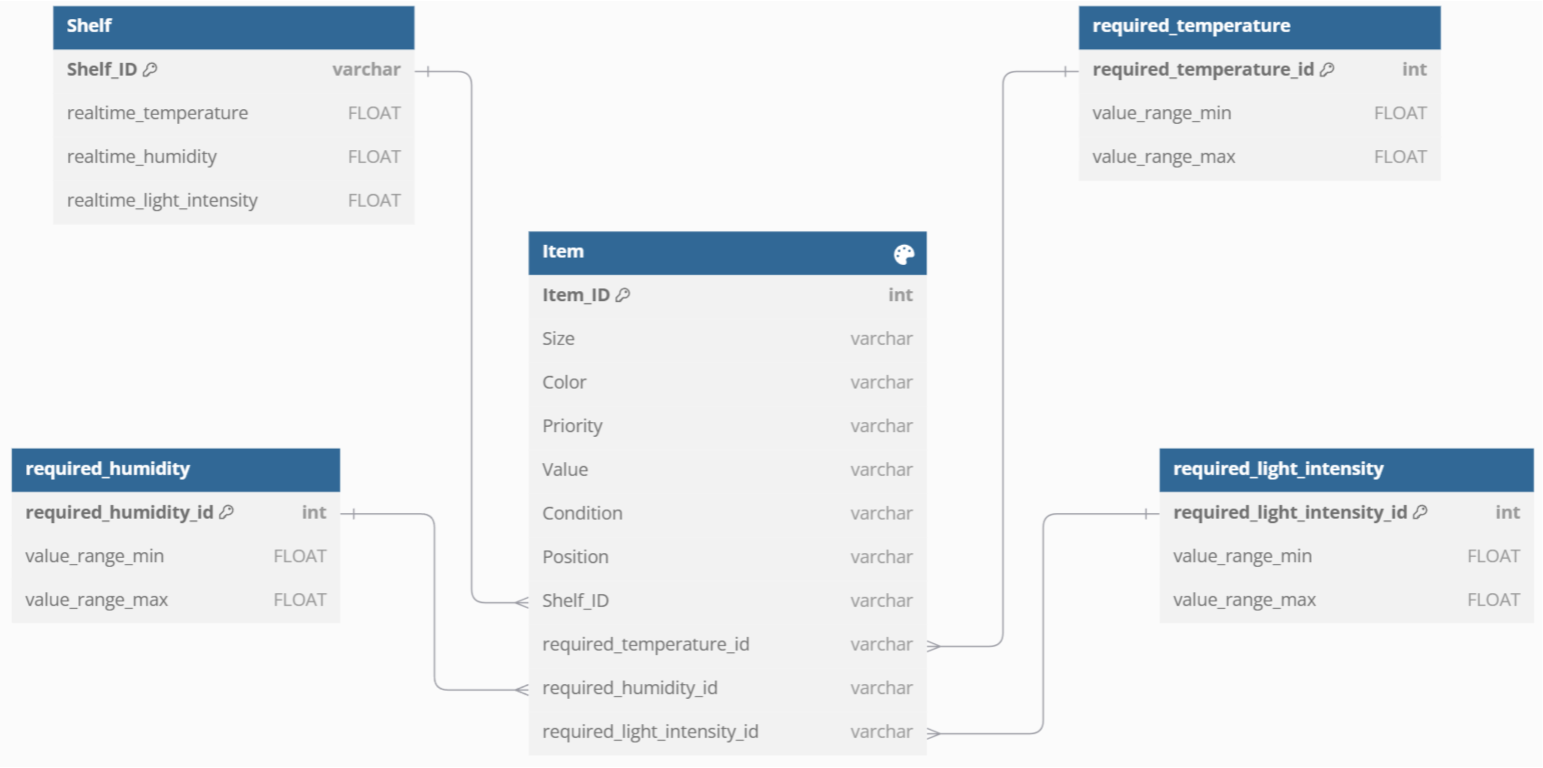
System Pipeline



1. Background
2. Basis
3. Conceptual Design
4. Implementation
5. Evaluation and Discussion
6. Outlook and Summary

Implementation

1. Database (defined in MySQL)



Implementation

2. Insert value in database.

Item	
Item_ID	int
Size	varchar
Color	varchar
Priority	varchar
Value	varchar
Condition	varchar
Position	varchar

required_light_intensity	
required_light_intensity_id	int
value_range_min	FLOAT
value_range_max	FLOAT

3. Supplement specific information in prompts.

Standard operation procedure	**1.** Inbound: 1) You get the information about the item to be stored and current available shelves from input. 2) You generate a function-call sequencen including pick and place, so that the item can be stored from other place to these shelves (use the name of the item(candy/cellphone/toy...) as its id) **2.** Outbound (…)

4. Coding.

```
def generate_task(user_input):
    with open(...) as file:
        task_generator_prompt = file.read()

    client = OpenAI(api_key=...)

    chat_completion = client.chat.completions.create(
        messages=[
            {"role": "user", "content": user_input},
            {"role": "system", "content": task_generator_prompt},
        ],
        model="gpt-4o",
    )
    result = chat_completion.choices[0].message.content
    return result
```

1. Background
2. Basis
3. Conceptual Design
4. Implementation
5. Evaluation and Discussion
6. Outlook and Summary

Evaluation

Test scenarios (with total 6 test cases, each case is executed 10 times)

Test with Data Query		Test without Data Query	
Test Case	**Text Input**	**Test Case**	**Text Input**
Storage	{"Store the watch"}	**Emergency Handling**	{“Fire alarm"}
Retrieval	{"Take out all items with high value"}	**Initialization**	{“Robotic arm Reset"}
Item organization	{"Rearrange items according to their value, high value first, then medium value at last low value"}	**Movement**	{“Go to next shelf"}

Evaluation

Storage

User's input	Success Rate (Test pass/Test total)	Typical bad case	Reason
1. {"Store the watch"}	0.5 (5/10)	The generated SQL query result: [] SQL query did not return results. Task planning skipped. Planned Task: None	SQL error
2. {"Store these two golden watches"}	0.2 (2/10)	error in your SQL syntax;	Asyntactic function calls， ambiguity
3. {"Store these two watches with high value, no specific requirement for temperature, humidity and light intensity"}	1.0 (10/10)	——	——
How to fix?	The **more specific** the input is and the **better** it matches the database content, the **easier** it is to accurately complete the task, while limiting semantic diversity.		

Evaluation

Retrieval

User's input	Success Rate	Typical bad case and Reason

1. {"Take out all items with high value"}	0.9	`The generated SQL query result:`
		`[]`
		`SQL query did not return results. Task planning skipped.`
		`Planned Task:`
		`None`
		Lack of necessary information

2. {"Take out all items with high priority and medium size, no specific requirement for temperature, humidity and light intensity"}	1.0	——
3. {"Take out all items with low priority, low value, small size and stored in the shelf with medium temperature, low humidity and low light intensity"}	1.0	——
4. {"Take out all items whose current shelves do not match their storage requirements (temperature, humidity and light intensity)"}	1.0	——

Evaluation

Items Organization

User’s input	Success Rate	Typical bad cases	Reason

1. {"Rearrange items according to their value, high value first, then medium value at last low value"}	0	`The generated SQL query result:`
		`[]`
		`SQL query did not return results. Task planning skipped.`
		`Planned Task:`
		`None`
			Data distortion

		`error in your SQL syntax;`	Asyntactic function calls

		`{`
		`"commands": [`
		`"pick (#item 01 from #shelf 02)",`
		`"place (#item 01 to #shelf 03)"`
		`]`
		`}`
		
		“real” control demand but actually not feasible
			Textual input not enough for high-complexity tasks

How to fix?	Data stored in a SQL database is not intuitive enough for tasks of high complexity. LLM needs a more “**visualized**” item distribution.		

Evaluation

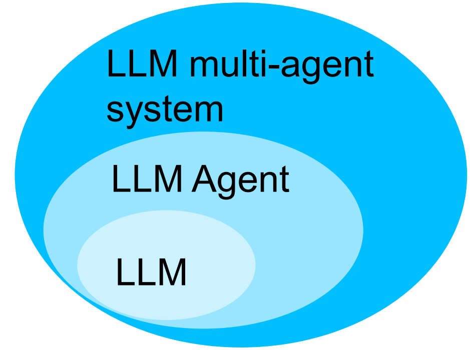
Tasks without Data Query

Test cases	User’s input	Success Rate
Emergency Handling	{“Fire alarm"}	1.0
Initialization	{“Robotic arm Reset"}	1.0
Movement	{“Go to next shelf"}	1.0
Result	LLM can handle these tasks with lower complexity.	

1. Background
2. Basis
3. Conceptual Design
4. Implementation
5. Evaluation and Discussion
6. **Outlook and Summary**

Summary and Outlook

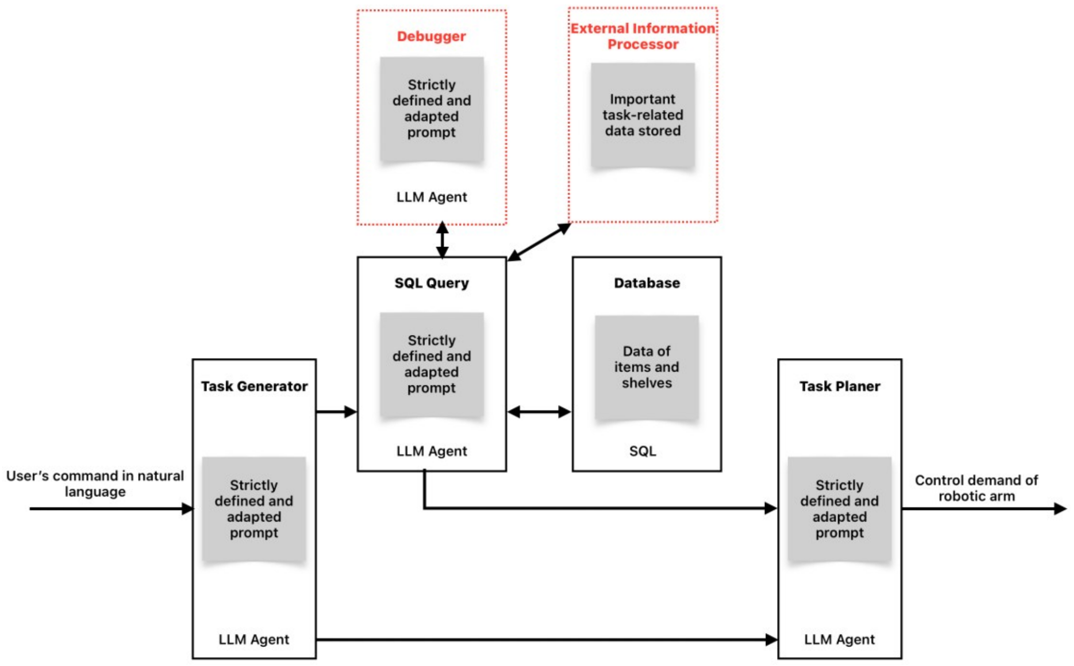
Summary



LLM based robotic arm
- LLM Agent
  - Task Generator
  - SQL Query Agent
  - Task Planer
- Database
- Debugger
- External Information Processor

Summary and Outlook

New System Pipeline



Potential SQL errors →

Iterative debugger

Data distortion →

External data store

Universität Stuttgart
Institute of Industrial Automation
and Software Engineering

Vielen Dank!

ZHE CAO

E-Mail st186915@stud.uni-stuttgart.de

Universität Stuttgart

Quelle

1. https://www.educba.com/what-is-neural-networks/

2. Y. Xia, N. Jazdi, and M. Weyrich, "Applying Large Language Models for Intelligent Industrial Automation: From Theory to Application: Towards Autonomous Systems with Large Language Models" Institute of Industrial Automation and Software Engineering, University of Stuttgart, 2023.