Twelfth Annual Advances in Cognitive Systems Poster Collection (2025) 117-131

Do as I Say, Not as I Do: The Role of LLMs in Open World Agents

Bill Ferguson BILL.FERGUSON@RTX.COM
Marshall Brinn MARSHALL.BRINN@RTX.COM
Allyson Beach ALLYSON.J.BEACH@RTX.COM
Aaron Adler AARON.ADLER@RTX.COM

Abstract

What is the right role for Large Language Models (LLMs) in agents? We describe here, results and observations from efforts to use LLMs as integral components of a procedure following cognitive agent architecture to be deployed in an open environment, guided by a human operator. Our agent starts with high-level tasking and the procedural knowledge to execute that tasking under normal circumstances. We will contrast this agent with a baseline agent that is a single, prompted LLM. In this simple role we find critical (and inherent?) limitations in the LLM's memory capacity: LLMs as agents can't consistently track plan execution state nor can they track changing world state (even when this state is clearly derivable from their context buffer). We do, however, find central roles in agent architectures where an LLM can provide significant value during procedure execution – updating procedures; translating representations; noticing, characterizing and adapting to anomalies; testing for conditions and action success; etc. We advocate for an architectural approach where programed components scaffold the execution state of the agent and help with context tracking, but where LLMs do most of the "reasoning". To both scaffold and exploit LLMs effectively, we also advocate a hybrid knowledge representation made up of formal frames or schema, many of whose slots are filled with natural language instead of symbolic structure. We speculate on then reasons that the limitations (and strengths) of LLMs may be inherent to the transformer architecture and the nature of LLM unsupervised training. We conclude that a likely role for LLMs in agentic systems is at the heart of each reasoning process in the agent but not as a stand-alone agent. Keywords: agent, agentic, LLM, plan execution, cognitive architecture, hybrid representation.

1 Introduction

Recent advances in Large Language Model (LLM) capabilities, along with advances and demand for autonomous deployed entities have raised the possibility that LLMs may be able to serve as the their reasoning engine and perhaps their entire execution control system. Our work reported here [1] is focused on precisely these questions: where can LLMs serve a productive role in an agent architecture and where might it be unadvisable? We limit ourselves to a (pseudo[2]) open world

[1] Work done under TCALLM (Teachable Cognitive Architecture using Large Language Models) a DARPA I2O project.
[2] The simulator we use is fairly simple and we have programed everything that can happen. We sometimes add in events and consequences that the agent's written instructions do not cover. In this limited sense the world is open. Of course, the LLM components probably have some competence at dealing with these new events and that is precisely what we hope to get from them.

where everything the agents sees or does is conveyed in natural language. In a closed environment, one can fully describe the procedure knowledge to achieve a specific set of goals. However, deployment to an open environment means the agent will inevitably confront surprises of different sorts: e.g. new instrutions, unexpected failures, and unfamiliar observations.

We therefore consider the full life cycle of the agent, and contributions an LLM may make in each of these:

- **Initialization**: Receipt of procedure knowledge, goals and tasking for the normal, expected case.
- **Normal Execution**: Execution of the procedure in the environment, invoking interfaces and processing responses in sequence.
- **Surprise Management**: Detecting that some interface response, an observation or instruction from a human Guide) not as expected and working to adapt the procedure in the short term to handle the surprise 'get back on track' (i.e. back to normal operations).
- **Procedure Refinement**: Detecting a pattern of surprises and responses that may constitute a 'new normal' and updating the standard procedure knowledge accordingly.

Figure 1 illustrates this life cycle of the agent in its initialization, normal execution, surprised and learning phases. In each phase, the agent must rely on and possibly modify procedure knowledge and world/execution state and potentially interact with the human guide.

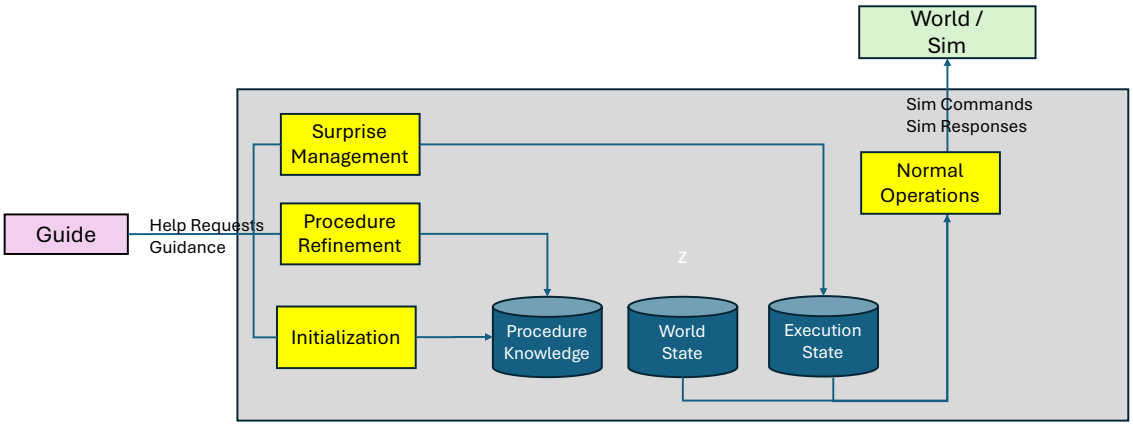


Figure 1. High-level Agent Architecture by Life-cycle Events

Our work pursues two different lines of research:

- **LLM Limitations in Maintaining State**; To what extent can an LLM maintain both world state and execution state?
- **LLM-enhanced Agent Architecture**: What kind of agent architecture can take advantage of the capabilities of LLMs while avoiding the limitations LLMs exhibit in an agent context?

2 Related Work

Agentic AI is of substantial current commercial interest and remains an interest to AI researchers. In the machine learning community, agentic AI has been mostly the domain of reinforcement learning but the generative model community has now expanded into the area as well. In this paper we are concerned with the appropriate role for LLM's in an agentic system and will focus on other work that relates to this.

An early venture into the use of LLMs in an agentic context is (Park et al 2023). In this inventive study the authors built a social simulation populated by agents that took actions and commuiniacted with each other in natural language, in a simulated community. The agents were able to accomplish social tasks such as arranging and participating in a party. Each agent used an LLM to generate text, choose actions and process the text produced by other agents. To use the LLM effectively, detailed prompts were created dynamically. These prompts told the LLM what was going on and what it needed to do. This is related to our notion of scaffolding but their work did not frame its LLM use in terms of cognitive functions.

AI digest's Agent village (2025) is an ongoing experiment where several public AI chatbots are connected into a group discussion and provided with goals. Each is given a virtual computer to work with. The village is open to human participants who may help or cause trouble. This environment goes beyond being open and often amount being antagonistic (meaning other agents are reacting to the test agents' actions and trying to confuse it). The results are mostly anecdotal but the agents seem to be losing track of their agendas (distracted by unrelated comments) or fixating on subgoals (getting applications or web services to run on their "virtual" computers) or even getting confused about their own capabilities (do they each have the own virtual computer (true) or are the sharing one common one (false)? This work is promising and we look forward to more good stories and ideas coming out of this project.

In (Wray, Kirk, & Laird, 2025) the authors explore the idea of universal, cognitve design paterns. Based on their prior work on SOAR and other architectures they start with paterns (Knowledge compilation, hustorical knowledge represenation) that are more complex than the fundamental ones, tracking world state and execution state, we consider here.

Ye Ye (2025) has similar agent building goals and a more complete system. Their work focuses on agents that achieve goals by developing plans and executing them with structural support for the execution process. They also use a hybrid representation called a task memory tree that they liken to an HTN. Their approach is orriented towards improving agent performance rather than a cognitive description of LLMs strenths and weaknesses.

Fabrizio (2023) posited the "jagged frontier" of LLM competence, claiming that LLM's (and multi-modal foundational models) are very good at some tasks but weak at others and that it is very hard (and remains an open research problem) to predict what will be inside the model's competence and what will be outside. This paper is a foray into mapping that fronter. We will not go the route of the formalists and try to determine a logical or mathematical delineation of the boundary, but, instead, we will try to answer in terms of classical cognitive architectures. We look at the aspects

of cognition that support plan execution and attempt to describe what an LLM can and can't to in those terms. This persepctive will also enable us to devise scaffolding to help a compound agent leverage LLM competencies and compensate for its weaknesses.

3 Limitations of LLMs Operating in Agentic Context

To successfully operate in a dynamic environment, an agent must possess strong memory skills of these specific kinds:

- **World state memory**: Given all the observations it has processed, what is the current state of the world?
- **Execution state memory**: Given the current procedure being executed and history of past steps executed, where are we in the current procedure and what is the next step to be taken?

Our contention is that LLMs, by themselves, are limited in their ability to perform these memory activities, particularly as the environment becomes more complex and the history becomes lengthier. This is not to argue that LLMs cannot support agentic behavior in dynamic environments; rather, it is to argue that LLMs must be supported by non-LLM software components that perform these tasks more reliably for an agent to perform reliably and robustly. To that end, we have run a set of experiments that seek to demonstrate these limitations in different ways, described in the following sub-sections.

3.1 LLM World State Memory Limitations

To test the ability of LLMs to accurately represent world state in a dynamic environment, we constructed a series of tests[3] of the following sort: Given a set of animals and a set of cities[4], we place the animals in the cities randomly. Then on each day we move random animals to random cities. We then ask about the locations of animals on different days.

Our first experiment entailed documenting these animal moves as "On day {day}, the {animal} was moved to {city}." We then asked about the locations of animals on the last day of this simulation. Figure 2 illustrates the degradation of the ability of an LLM to correctly identify the ultimate location of animals after many moves.

[3] We performed these tests summarizing results over both OpenAI gpt-41 and Anthropic claude-3.5 models.

[4] We started with numbered boxes and balls but found that by using named animal types and cities, the models did better (and since we are demonstrating weakness, we wanted them to do as well as they could). Perhaps animals and cites worked better because they had a more elaborated structure in the LLM's network.

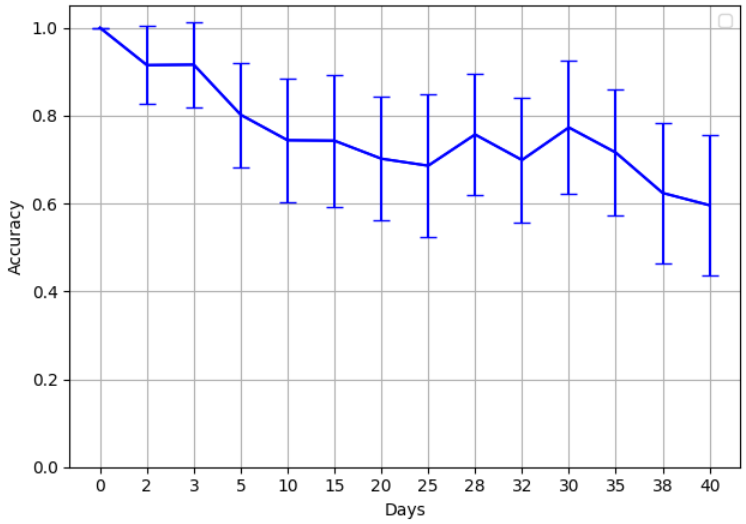


Figure 2. Ability to identify final location of animals after many days of moves.

To push this further we wanted to simulate unobservable world state that must be retained. We modified the way the moves were reported to the LLM, replacing animal moves with city swaps. That is, we documented the moves as 'On day {day}, the animals in {city1} were swapped with the animals of {city2}." This requires the LLM to track what animal is in what city (or rededuce it from their conext memory) to answer quieries. Figure 3 illustrates the degradation of the ability of an LLM to correctly identify the ultimate location of animals after many swaps.

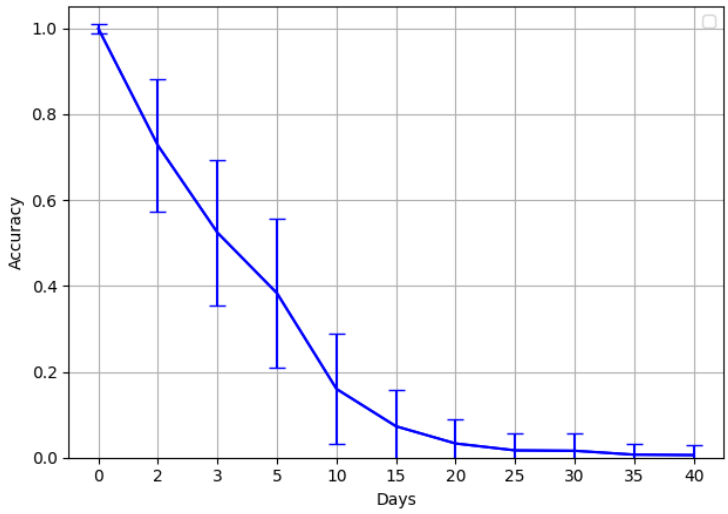


Figure 3. Ability to identify final location of animals after many days of swaps.

3.2 LLM Execution Memory Limitations

We constructed a set of experiments in which an LLM was provided a given procedure was provided in JSON format and a history of the past steps taken within the procedure. The procedures were manually generated with varying complexity in their tree structures. We then asked the LLM to identify the next step through the course of the simulated execution of the procedure and measured the error rate.

Figure 4 illustrates the degradation of proper identification of next step as procedure knowledge becomes more complex. ARCH0 represents an architecture in which the agent relies on an LLM to maintain a sense of current execution context (stack and next step); ARCH1 represents an architecture in which the agent uses a coded component to maintain a sense of the current execution context..

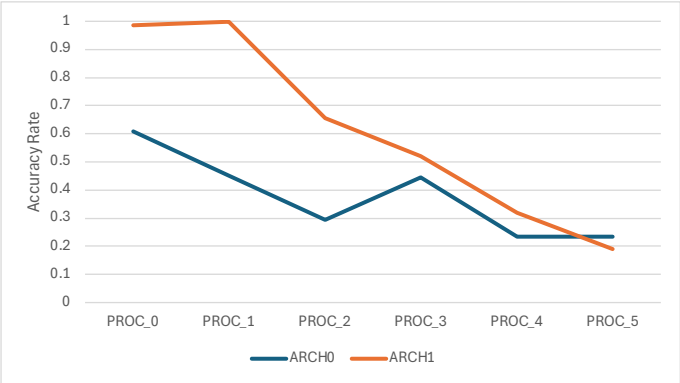
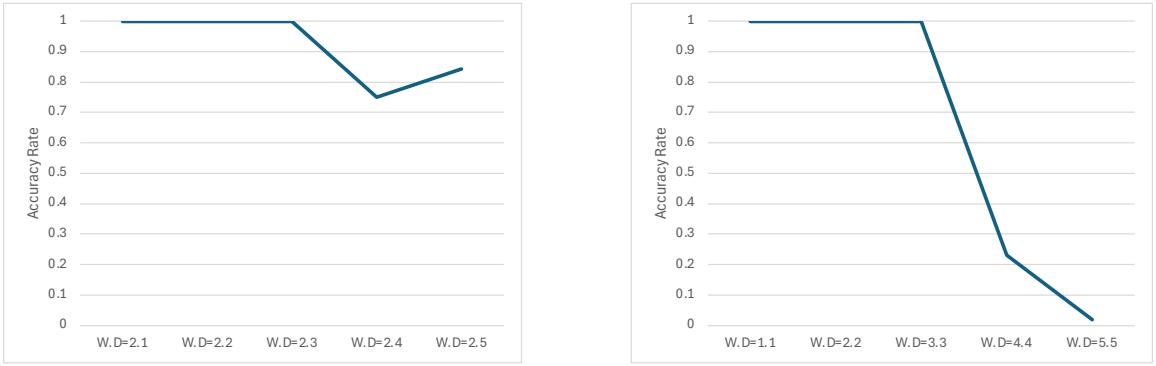


Figure 4. Next Step Error Rate as a Function of Problem Complexity.

In addition, we automatically generated procedure trees of different widths and depth and asked an LLM to not only specify the next step but the current execution stack (i.e. what are all the procedure nodes we are currently executing?). The test reflected a declining ability of the LLM to correctly identify the current stack as the procedure became more complex.

Figure 5 illustrates the degradation of proper maintenance of the execution stack as procedures become more complex, where “W” and “D” represent the width and depth of the procedure tree structure.



For the same width and increasing depth, it was perfect for width = 2, depth=1,2,3 but deteriorated a little for depth=4, 5.

For the same width and depth, it was fine for 1,1 2,2 and 3,3 and then fell apart for 4,4 and 5,5.

Figure 5. Ability of an LLM to determine the correct stack for procedures of increasing complexity.

3.3 Managing Multiple Execution States

To evaluate the ability of the LLM to follow instructions in the context of a dynamic state, we designed a series of tests based on a "manager–employee" structure. In this setup, the LLM acts as a manager assigning tasks to a set of employees. The tasks are derived from a predefined procedure, an ordered list of instructions.

For example, a procedure might be: ["unlock the door", "open the door", "sweep the floor", …]. In such a case, if an employee completes "unlock the door," the manager should assign the next task in sequence, "open the door."

In each test, the employee requesting the next task is chosen randomly. This results in employees progressing at different rates, requiring the manager to track the state of each employee in real time. At the start of the test, the LLM is given the complete procedure. In the background, a "truth state" of each employee is tracked and compared with the state inferred by the LLM. Accuracy is measured as the ratio of correct assignment given from the LLM over the total number of assignments.

We ran experiments in two different information contexts:

- **First Condition** – State Provided. The employee provides both their ID and the last task completed when requesting the next task. This allows the LLM to look up the employee's progress without maintaining state internally. As shown by the blue line in Figure 6, the LLM achieves perfect accuracy (100%) under this condition, with zero standard deviation.
- **Second Condition** – State Not Provided. In the next test, the employee provides only their ID, omitting the last completed task. Here, the LLM must track each employee's progress solely based on the tasks it previously assigned. As shown by the green line in Figure 6, performance declines as the number of employees increases. Even with only three employees, accuracy is degraded due to the complexity of tracking multiple independent states.

3.4 Summary of LLM Limitations Experiments

These experiments represent a small sample of the kinds of experiments one could run on LLMs to assess their ability to maintain world and execution state. But the results are suggestive. The ability of LLMs to clearly determine the state of objects in the world based on many statements about changes to the state of many objects is limited and degrades as the simulated world grows in complexity. Cognitive hints do not seem to provide much added benefit, though there is more research to be done in this area. Further, the ability of LLMs to maintain a sense of what it is doing (where in the procedure stack, what step to do next) degrades as a function of procedure complexity as well. These results motivate an approach towards building agents wherein we try to leverage the strengths of LLMs while hand-coding components to do those tasks where LLMs are less reliable.

4 Procedure Knowledge Representation

We shift focus now to exploring agent architecture approaches to leverage LLM capabilities while not being constrained by LLM limitations described below. To that end, we have built an experimental domain and simulator and an agent framework for investigating how LLMs can be used to support different phases of the agent life-cycle.

TCALLM Domain. For our work, we have adopted a domain of laboratory cleaning. This domain has many routine procedures but also offers many potential surprises, such as:

- *Unusual Room Conditions*: The room may report some unusual condition that may need addressing directly or may interfere with normal operations, for example a flooded floor.
- *Missing Equipment*: The room may require some equipment (e.g. a mop) not ordinarily carried as part of normal routine.
- *Notes*: A note on the door or whiteboard may provide additional guidance from the users of the lab (e.g. "Don't erase the whiteboard" or "Please leave the door unlocked").

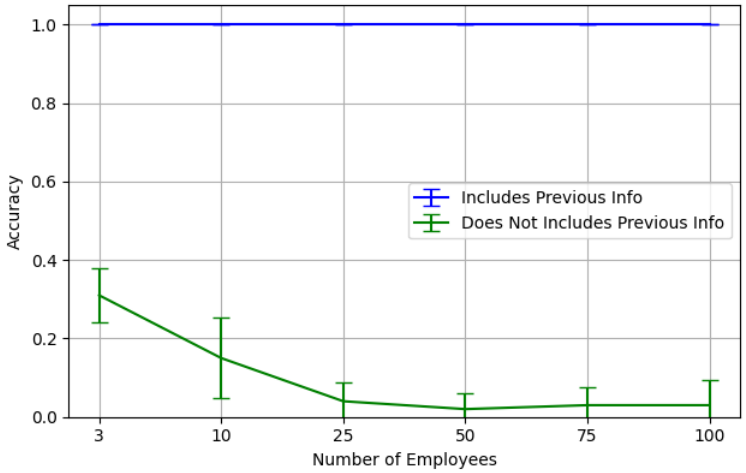


Figure 6. Accuracy of Determining Next Task for a series of employees with and without indicating previous task.

We build a simulated environment for this domain representing the state of each room in the building with scripted surprises. The simulator takes commands in English and responds with results/acknowledgement in English.

Figure 7 show the list of interface commands available for interacting with the TCALLM Laboratory cleaning simulation, along with descriptions and, where appropriate, input and output argument types.

```
class CleanerCommands(Enum):
   ENTER_BUILDING = ("Enter building at start of work day")
   EXIT_BUILDING = ("Exit building at end of work day")
   CHANGE_FLOOR = ("Change the current floor", ["The number of current floor"], [int])
   CHANGE_ROOM = ("Go to door of current room on current floor", ["the number of the current room"], [int])
   ENTER_ROOM = ("Enter the room you are facing")
   EXIT_ROOM = ("Exit the room you are in")
   OPEN_DOOR = ("Open the door of the room you are facing")
   CLOSE_DOOR = ("Close the door of the room you are facing")
   UNLOCK_DOOR = ("Unlock a locked door you are facing")
   LOCK_DOOR = ("Lock a locked door you are facing")
   CLEAN_WHITEBOARD = ("Clean the whiteboard in the room you are in")
   EMPTY_TRASH = ("Empty trash in can in room you are in")
   PICKUP_TRASH = ("Pickup and dispose of trash on the floor of the room you are in")
   VACUUM_CARPET = ("Vacuum carpet in the room you are in")
   ORIENTATION_INFO = ("Get back the current orientation info of the agent: what room, what floor, inside")
   DOOR_INFO = ("Get the state of the door: whether it is open, closed or locked", None, None,  "domain.building.CleanerStateEnums.RoomDoorState")
   ROOM_INFO = ("Get the state of the room including time last cleaned", None, None, "domain.building.CleanerStateEnums.RoomCompositeState")
   BUILDING_INFO = ("Get list of floors and rooms per floor")
   FLOOR_LIST = ("Get list of floors in building", None, None, list)
   ROOM_LIST = ("Get list of rooms on given floor", None, None, list)
   SIGN_IN = ("Sign in for starting cleaning work for the day", ["The current day"], [int])
   SIGN_OUT = ("Sign out for ending cleaning for the day", ["The current day"], [int])
```

Figure 7. TCALLM Laboratory Cleaning Interface Commands.

Hybrid Representation. We have chosen a representation for procedure knowledge as hybrid of structured (JSON) and unstructured (English). Specifically, the JSON format consists of nodes with the following fields:

- **Name**: the name of the procedure node
- **Description**: English description of the intent of the node
- **Inputs**: List of input variables set in the scope of this node
- **Outputs**: Variables set with the results of this node.
- **Type**: Whether the node is f type ‘command’ (an explicit interface command to invoke) or ‘loop’ (a loop over a given input variable).
- **Command**: The interface command to invoke.
- **Preconditions**: A list of English phrases to be evaluated as gates to executing the given node.

Figure 8 shows an example TCALLM Agent procedure in executable JSON format.

```
{ "name" : "CleanBldg", "description" : "Here's the procedure for cleaning the building.",
  "children" : [
    {"name" : "SignIn1", "description" : "Sign in on day 1."},
    {"name" : "FloorList", "description" : "Get list of floors in building", "inputs" : [], "output" : "ALL_FLOORS"},
    {"name" : "CleanEachFloor", "description" : "Clean each floor", "type" : "LOOP",  "inputs" : ["ALL_FLOORS"], "output" : "CURRENT_FLOOR", "children" : [
  {"name" : "GoFloor", "description" : "Go to floor", "inputs" : ["CURRENT_FLOOR"], "children" : [
    {"name" : "RoomList", "description" : "Get list of rooms on current floor", "inputs" : ["CURRENT_FLOOR"], "output" : "ALL_ROOMS_ON_FLOOR"},
    {"name" : "CleanEachRoom", "description" : "Clean each room in list of rooms", "type" : "LOOP", "inputs" : ["ALL_ROOMS_ON_FLOOR"], "output" : "CURRENT_ROOM", "children" : [
        {"name" : "PrepTasks", "description" : "Perform preparatory tasks", "children" : [
       {"name" : "GoRoom", "description" : "Go to room.", "inputs" : ["CURRENT_ROOM"]},
       {"name" : "UnlockDoor", "description" : "Unlock the door.", "preconditions" : ["Door is locked"]},
       {"name" : "OpenDoor", "description" : "Open the door."},
       {"name" : "EnterRoom", "description" : "Enter the room."}
  ]},
  {"name" : "CleanTasks", "description" : "Perform cleaning tasks", "children" : [
  {"name" : "PickupTrash", "description" : "Pick up any trash"},
  {"name" : "VacuumFloor", "description" : "Vacuum the floor"},
  {"name" : "EmptyTrash", "description" : "Empty the trash bin"},
  {"name" : "EraseWhiteboard", "description" : "Erase the whiteboard"}
  ]},
  {"name" : "FinalizeTasks", "description" : "Perform final tasks", "children" : [
  {"name" : "LeaveRoom", "description" : "Leave the room."},
  {"name" : "CloseDoor", "description" : "Close the door."},
  {"name" : "LockDoor", "description" : "Lock the door."}
  ]}
  ]}
  ]}
  ]},
    {"name" : "SignOut1", "description" : "Sign out on day 1."}
  ]
}
```

Figure 8. Example TCALLM Agent Procedure Knowledge Representation.

The nature of the hybrid representation allows for the agent to use LLMs as needed. Because it has a defiend struture, the coded parts of the agent can manipulate it (e. g. procedure tracking); because it holds free text it can "represent" potenatilly much more of what it may encounter in an open world. The agent's coded control struture can use the LLM to evaluate the preconditions in the current context. If no 'command' is provided, it can use the node name and description to determine the best interface command to achieve the intent of the node.

5 LLMs Support for Execution

We have explored the following mechanisms for leveraging LLMs in the 'normal' (unsurprising) agent execution of procedures.

Preconditions and Context Memory. A known challenge with LLMs is that they are often confused by multiple statements that are given in different contexts. When evaluating a precondition, for example, "If the door is locked", the LLM may be asked to evaluate this phrase in the context of many statements from previous rooms or doors regarding the state of doors being locked or unlocked. For that reason, we maintain a 'context memory' in which all responses from the simulator (statements like 'The door is unlocked') are tagged with current context (e.g. day, floor, room). The dimensions of the context may be determined by the LLM, the JSON structure or by human guidance. Given this context, the LLM can be given only the relevant previous statements about door status for the current context in attempting to evaluate a precondition.

Agent Initialization. In a closed world (where all eventualities can be anticipated) an agent is typically fielded with programed or learned procedures for achieving its goals. In TCALLM, we allow for the procedure knowledge to be provided in English and translated by an LLM into the executable JSON format given above. Our experience to date is that the LLM can reliably handle

fairly complex procedures, including both monolithic procedures and procedures that references other existing procedures. As will be noted later, the limits on the scaling of this capability as a function of procedure complexity warrants further exploration. (Of course in an open world, this procedure will be inadequate. See the next sections for how LLMs help us cope with this.)

Figure 9 shows an example of NL provided description of a procedure, and the LLM-generated representation capturing that procedure in executable JSON format.

Input Lesson

```
In this lesson you're going to learn a new skill called BuildingClean.
It will describe what to do to clean a building.
It assumes you are given the current day.
Here are the steps:
Sign in for the current day.
Ask for a list of floors in the building.
Perform FloorClean for each floor in this list.
Sign out for the current day.
```

Parsed Procedure

```
{ "name" : "BuildingClean", "description" : "What to do to clean a building", "children" : [
    {"name" : "Sign in", "inputs" : ["current_day"], "type" : "COMMAND", "command" : "SIGN_IN"},
    {"name" : "Get List of Floors", "output" : "FLOOR_LIST", "type" : "COMMAND", "command" : "FLOOR_LIST"},
    {"name" : "Clean Each Floor", "inputs" ["FLOOR_LIST"], "type" : "LOOP", "command" : "FloorClean"},
    {"name" : "Sign out", "inputs" : ["current_day"], "type" : "COMMAND", "command" : "SIGN_OUT"},
    ]
}
```

Figure 9. Example TCALLM Lessons: BuildingClean

6 LLM Support for Surprise Management

Execution of any fixed procedure in an open environment will inevitably result in surprise, for any of a variety of reasons, such as:

- The world doesn't respond to actions as expected, either due to a sensor failure, actuation failure, or the world not behaving as expected.
- The world provides information that is qualitatively different from expectations (i.e. contradicts the assumptions built into the model) or ontologically different (i.e. encounter a new concept or object not built into the original ontology).
- The goals of the task are changed by the guide or by the need to deal with unexpected circumstances.

A robust agent in an open environment must be able to continue to try to pursue its goals even in the face of such surprises. Ideally it should be able to take advantage of these new circumstances; at the minimum it should not fail while it tries to weather the surprise and get back to an expected environment.

Our work in TCALLM has explored several different approaches whereby LLMs play a powerful role in surprise management.

Guidance Modification. The human guide may interrupt the agent at any time with a change to priorities or policy. But rather than providing an entirely new procedure, the guide may ask the agent to make some temporary excursion, not changing the underlying procedure but changing the knowledge in some specific context. For example, the guidance may come in the "Evacuate the Building Immediately" or "Don't bother with the 3rd floor today". Both examples require the agent to modify its behavior but not change its general understanding of the procedure to be executed.

In these cases, we have configured an LLM to take the guidance and compile it into an executable 'patch' to the procedure that only applies to the current (or otherwise specified) context. This is challenging because the agent may be many layers deep in some procedure stack and must modify the stack as it is being executed. The LLM is provided with the current procedure and stack information and the guidance and provides a JSON structure that the infrastructure can execute on a temporary basis.

Surprise Recognition. Being surprised entails finding meaningful differences between expectation and observation. Beyond the establishment of expectation, we need a reasonable approach towards 'meaningful differences' to avoid continually being surprised by small or meaningless differences. We have found that the LLM is useful in providing the nature of the action, the context, the expectation and the response and asking, "Is there a significant and relevant difference between expectation and observation?". The LLM is often able to handle soft concepts like 'significant' and 'relevant' more reliably that a hard-coded set of criteria, particularly since those criteria may not have anticipated the surprise when written.

Surprise Response. Response to surprise, that is, selecting the proper sequence of activities to address or mitigate the surprise, is a planning problem. LLMs have shown that they are competent at simple planning problems but become unreliable as problems become more complex. What we have focused on in TCALLM is using the Agent/Guide dialog to discuss possible approaches towards surprise response:

- *Human Guided*: The Agent reports a description of the surprise (including context, expectation and observation) and the Guide responds with English guidance, not unlike the 'patch' approach described above.
- *Agent/Guide dialog*: We have experimented with the Agent performing some planning and offering a solution to the Guide for critique. The Guide can then provide a full English response or provide comments on the Agent's proposed plan from which the Agent will re-plan if possible.
- *Agent Ensemble*: We are beginning to explore the value of ensembles of LLMs for generating possible surprise responses. We set up a blackboard architecture and several LLMs with different roles (e.g. the Brainstormer, the Critic, the Cost Assessor, the Feasibility Analyst, etc.) and let these LLMs iterate until a candidate solution is derived. From there we have the Agent send the candidate solution to the human guide for approval or feedback.

7 LLM Support for Procedure Adaptation

Over the course of a long performance period, an agent will encounter many different surprises, and sometimes, many instances of similar surprises. One attribute of a successful agent is to enhance its procedural knowledge over time so that what was once surprising is later treated as routine. Such a capability provides benefits of two kinds of efficiency: 1) surprise detection and

response is typically more costly than the standard execution of procedures, and 2) reworking the procedure to include a new case provides the opportunity to optimize the procedure itself.

This plan refinement is a kind of learning, but not one in which the LLM itself is trained. Rather the LLM along with the human Guide, learn enhanced procedural knowledge over time. The challenge of this kind of learning is that it rests on many complex judgements that are not amenable to explication in symbolic language:

- How many instances of a surprise are enough to justify calling it a pattern?
- What are the common elements of these pattern events, and what are the irrelevant details?
- How similar do the instances need to be to make them be considered parts of the same underlying issue and handled by the same underlying response?

In our experience in TCALLM, LLMs are well suited to handle these kinds of questions. While we do not advocate LLMs making decisions to modify procedure knowledge autonomously, we do see the value of LLMs autonomously analyzing the history of previous surprises, inferring patterns and suggesting plan augmentations to the human Guide for modification or approval.

Figure 10 shows an example of an LLM taking a log of past surprises and responses and proposing a change handle certain surprises as part of the base procedure.

Log of surprise events and responses in context.


```
[
 {"context" : {"day" : 1, "floor" :  1, "room" :  7}, "surprise" :  "Couldn't clean room: there was a puddle on the floor",
  "response" :  "Went to supply closet. Took the mop and bucket. Returned to room. Mopped up puddle. Returned mop and bucket to supply closet. Returned to room"},
 {"context" : {"day" : 1, "floor" :  2, "room" :  3}, "surprise" :  "Couldn't clean room: there was a puddle on the floor",
  "response" :  "Went to supply closet. Took the mop and bucket. Returned to room. Mopped up puddle. Returned mop and bucket to supply closet. Returned to room"},
 {"context" : {"day" : 2, "floor" :  1, "room" :  2}, "surprise" :  "Couldn't clean room: there was a puddle on the floor",
  "response" :  "Went to supply closet. Took the mop and bucket. Returned to room. Mopped up puddle. Returned mop and bucket to supply closet. Returned to room"},
 {"context" : {"day" : 2, "floor" :  3, "room" :  1}, "surprise" :  "No trash bin in room",
  "response" :  "Went to room next door. Took trash bin. Returned to room. Picked up trash. Returned trash bin to room next door. Returned to room."},
 {"context" : {"day" : 3, "floor" :  3, "room" :  4}, "surprise" :  "Couldn't clean room: there was a puddle on the floor",
  "response" :  "Went to supply closet. Took the mop and bucket. Returned to room. Mopped up puddle. Returned mop and bucket to supply closet. Returned to room"},
 {"context" : {"day" : 4, "floor" :  2, "room" :  5}, "surprise" :  "Couldn't clean room: there was a puddle on the floor",
  "response" :  "Went to supply closet. Took the mop and bucket. Returned to room. Mopped up puddle. Returned mop and bucket to supply closet. Returned to room"}
]
```


LLM provided pattern for potential procedure refinement:

```
RESP(Refine): ['I recommend adding a routine step to check for puddles
on the floor during room preparation and, if one is found, fetch and
use a mop and bucket from the supply closet before beginning regular
cleaning tasks.']
```

Figure 10. Proposed Procedure Refinement.

8 Future Research

Before our work can be applied to practical problems or to the guidance of training LLM's to be better agents, we need results that are more robust and cover a wider range of phenomena and scaffolding approaches. We should explore:

- Additional characterization of limits of LLM on world and execution state.
- More exploration of the effects of prompting on these weaknesses.
- Exploration of which tasks of procedure maintenance and adaptation can be done autonomously by an LLM, and which should be done in collaboration with the guide.
- Limitations of complexity of procedures that can be edited, patched.
- Optimization of proposed procedure refinements.
- Fine tuning LLMs on simulation traces as well as data like those in sections 3.1 and 3.2 to see if a model's attention mechanisms can be trained to better support procedure execution.

9 Discussion

To better understand and characterize the strengths and limitations of foundation models, we may need a more fundamental definition of agentic competence. Certainly, it is hard to imagine an agent being competent if it can't track what it is trying to do (it must be able to execute a plan). Also, an agent needs to remember and accurately track information that it has acquired during its goal pursuit to function in partially observable worlds. Other cognitive capacities (such as long-term memory and meta cognition) are very valuable, especially if the agent is collaborating, but the first two seem more fundamental and we have argued that they represent deficits in LLM competence.

LLMs are expert authors and composers, demonstrating some common sense; deep and unimaginably wide knowledge; and expertise at crafting both prose and code (as well as hybrids of the two). They can design and build things when those things are built out of strings of characters. Perhaps this is unsurprising since most of their training data is human composed artifacts (language and code)[5]. So, they become composers by mass imitation, which has (surprisingly, to many AI researchers) enabled profound competence as deep imitation has given rise to interpolation and some real degree of generalization. Yet we have no reason to believe that LLM's encounter more than a peripheral exposure to and need for (e. g. a learning signal) the cognition of getting things done by acting in the world. Perhaps, foundation models cannot yet approximate the cognition of execution because they have not been exposed to it at scale.

We have presented quantitative and anecdotal evidence that LLMs (at least at their current form) cannot act as agents on their own. Furthermore, we (and others) have presented mechanisms whereby LLMs can be scaffolded with plan following and relevance filtering mechanisms so that they can function more effectively. While LLMs are not good plan executers, they are good plan makers and they have some (under explored) capability to notice, explain and adapt to surprises. We therefore advocate central roles for them into the creation effective cognitive agents when used with execution scaffolding and hybrid knowledge representation.

[5] And now, also text written by other LLMs which may complicate this reasoning but since this new text is an imitation of human text, the observation may still hold.

References

Park, J. S., O'Brien, J., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). Generative Agents: Interactive Simulacra of Human Behavior. Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, 1–22. https://doi.org/10.1145/3586183.3606763

The AI Village (https://theaidigest.org/village) created by The AI Digest (theaidigest.org), in turn created by SAGE (sage-future.org), specifically https://theaidigest.org/village/blog/season-recap-agents-raise-2k (2025)

Smith, J. (2024, July 15). The future of AI in education. Education Insights. https://www.educationinsights.com/blog/future-of-ai

Fabrizio, D., McFowland, E., Mollick, E., Lifshitz-Assaf, H., Kellogg, K., Rajendran, S., Krayer, L., Candelon, F., and Karim Lakhani, K., (2023). Navigating the Jagged Technological Frontier: Field Experimental Evidence of the Effects of AI on Knowledge Worker Productivity and Quality. Harvard Business School Working Paper, No. 24-013

Wray, R. E., Kirk, J. R., & Laird, J. E. (2025). Applying Cognitive Design Patterns to General LLM Agents (No. arXiv:2505.07087). arXiv. https://doi.org/10.48550/arXiv.2505.07087

Wray, R. E., Kirk, J. R., & Laird, J. E. (2025). *Applying Cognitive Design Patterns to General LLM Agents* (No. arXiv:2505.07087). arXiv. https://doi.org/10.48550/arXiv.2505.07087

Ye, Y. (2025). Task Memory Engine (TME): A Structured Memory Framework with Graph-Aware Extensions for Multi-Step LLM Agent Tasks (No. arXiv:2504.08525). arXiv. https://doi.org/10.48550/arXiv.2504.08525