

Comparing LLMs for Prompt-Enhanced ACT-R and Soar Model Development: A Case Study in Cognitive Simulation

Siyu Wu¹, Rodrigo F. Souza², Frank E. Ritter¹, Walter T. Lima Jr²

¹The Pennsylvania State University - Penn State — USA

²Federal University of Sao Paulo - UNIFESP — Brazil

sfw5621@psu.edu — rodrigo.ferreira@unifesp.br

frank.ritter@psu.edu — walter.lima@unifesp.br

Abstract

This paper presents experiments on using ChatGPT4 and Google Bard to create ACT-R and Soar models. The study involves two simulated cognitive tasks, where ChatGPT4 and Google Bard (Large Language Models, LLMs) serve as conversational interfaces within the ACT-R and Soar framework development environments. The first task involves creating an intelligent driving model using ACT-R with motor and perceptual behavior and can further interact with an unmodified interface. The second task evaluates the development of educational skills using Soar. Prompts were designed to represent cognitive operations and actions, including providing context, asking perception-related questions, decision-making scenarios, and evaluating the system's responses, and they were iteratively refined based on model behavior evaluation. Results demonstrate the potential of using LLMs to serve as interactive interfaces to develop ACT-R and Soar models within a human-in-the-loop model development process. We documented the mistakes LLMs made during this integration and provided corresponding resolutions when adopting this modeling approach. Furthermore, we presented a framework of prompt patterns that maximizes LLMs interaction for artificial cognitive architectures.

Introduction

The study of human cognition and decision-making processes has been a longstanding pursuit in cognitive science and artificial intelligence research. Two prominent frameworks for cognitive modeling are the ACT-R (Anderson 2009) and Soar (Laird 2019).

These frameworks are powerful tools for simulating human behavior in various cognitive tasks. However, traditional model development for ACT-R and Soar involves complex coding and challenges in acquiring sufficient data for model refinement, hindering widespread adoption and accessibility. It has been noted as difficult by several authors (Pew and Mavor 1998; Ritter, Kukreja, and Amant 2007; Shakir 2002). Additionally, integrating perceptual and motor behaviors into either architecture's models presents challenges, limiting their applicability in real-world scenarios.

Recent advancements in language models like ChatGPT4 and Google Bard have opened new possibilities for enhancing ACT-R and Soar model development. We refer to

them here together as Large Language Models (LLMs) (Cerf 2023). ChatGPT4 (OpenAI 2023), a next-generation conversational AI, demonstrates remarkable proficiency in generating human-like responses across various tasks, and Bard (Google 2023) excels in providing diverse options. LLMs have significant potential in fields where humans and AI tools collaborate as dependable and reliable partners, resulting in improved developments of software-dependent systems (White et al. 2023), including artificial cognitive architectures. Nonetheless, the lack of existing research using LLMs in ACT-R and Soar model development amplifies the limitations of this approach.

This article focuses on building ACT-R and Soar models for two scenarios. In the first scenario, the ACT-R framework is used to develop a model that has the potential to interact with an unmodified interface, incorporating visual and manual modules. Specifically, the simulation involves starting a bus and maneuvering it on the road (Wu et al. 2023). However, the ACT-R model part does not delve into the integration of visual and motor management systems, like Segman (Ritter, Kukreja, and Amant 2007), Jsegman (Schwartz, Tehranchi, and Ritter 2020), or VisiTor (Tehranchi, Bagherzadeh, and Ritter 2023) to access the unmodified interface, as it falls beyond the scope of this article.

In the second scenario, Soar is used in an educational simulation to identify the dominant intelligence type of student. The simulation analyzes data from educational activities, such as solving mathematical problems and performing demanding tasks (Huizinga, Baeyens, and Burack 2018), to determine the student's dominant intelligence type according to Gardner's (Gardner 1993) theory of multiple intelligences.

By leveraging the language generation capabilities of LLMs, this article provides a broader understanding of the applicability and potential of these approaches in different contexts. The results of this experiment contribute to the advancement of research in artificial cognitive systems and offer valuable perspectives for improving intelligent systems in future practical applications.

Theoretical Foundations

Before describing the models, we briefly introduce the two architectures.

ACT-R

ACT-R is a cognitive architecture and a theory of simulating and understanding human cognition (Anderson 2009) (Ritter, Tehranchi, and Oury 2019). Its theory is embodied in the ACT-R software, through which we can construct models that can store, retrieve, and process knowledge, as well as explain and predict performance (Bothell 2017).

There are currently two kinds of knowledge representations in ACT-R, and they are declarative knowledge and procedural knowledge. Declarative knowledge consists of chunks of memory (e.g., apple is a kind of fruit), while procedural knowledge performs basic operations, moves data among buffers, and identifies the next instructions to be executed (e.g., to submit your answer, you have to click submit bottom). When the model is driving a bus in first-person perspective, these pieces of information will contain information such as what visual items presented to look at and what tasks to do next.

Soar

Soar has its origins in the groundbreaking work done by Newell and Simon around the 1950s through the mid-1970s, also inspired by the "General Problem Solver" created by George Ernst and Newell. While ACT-R was designed to model human behavior, Soar was inspired by it. Current understanding and hypotheses regarding cognitive architecture are incorporated into Soar 9, which has been in development for over 30 years and continues to evolve gradually. Soar's general computing concept is based on: objectives, problem spaces, states and operators (Laird 2019) (Newell 1990).

Model Development

We adopt an iterative design and development process that incorporates the concept of prompt engineering (White et al. 2023). This involves crafting prompts tailored from a Prompt Engineering GitHub repository (GitHub 2023) to emulate cognitive operations and actions, including providing context, posing perception-related queries, simulating decision-making scenarios, and evaluating the system's responses. Moreover, our prompts are designed with the integration of output customization (White et al. 2023) in mind, which emphasizes restricting or adapting the types, formats, structure of the output produced by the LLMs. We used personas to enhance the LLMs' situational awareness and provided recipes for LLMs, which include a sequence of steps or actions to achieve a specific end result, while also adopting a human-in-the-loop approach when faced with partial information or constraints.

Throughout the development, we fine-tuned the prompts based on the evaluation of the model's behavior. The careful engineering of prompts plays a pivotal role in improving the quality and accuracy of the model's responses. By refining the prompts, we aim to use LLMs to create ACT-R and Soar models.

ACT-R Model

The simulation task involves the model looking around a virtual environment and checking for the presence of

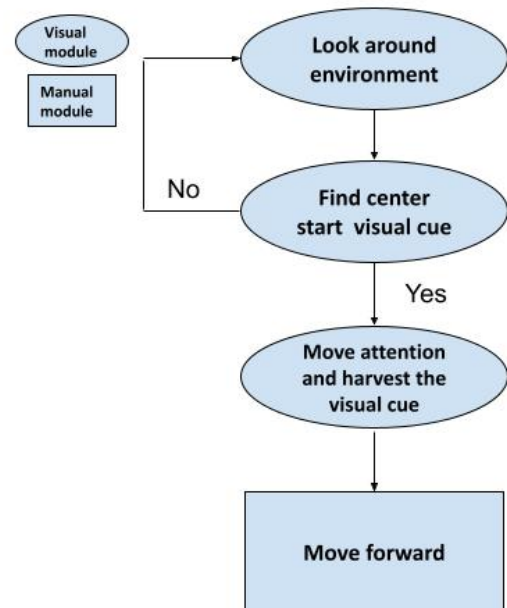


Figure 1: Flow of control in the ACT-R model to drive the bus.

a visual cue to start the bus. The environment, a video game called "Drive the Bus", is downloadable from Steam (<http://bit.ly/desert-bus-vr>) for Windows, was unaltered for the model. We create a simple flow chart to indicate the basic process the model will follow (see Figure 1).

Provide Context and Prompt for Declarative Chunk(s)

To initiate the model development, the first prompt outlined the simulation context and requirements, asking for recommended chunk types for the model. The prompt is designed as follows: "act as an ACT-R modeler to build a model, which name is DriveBus. the simulation environment is the video game "Desert Bus VR". The model is designed to look around the simulation environment, and if the model finds the center lane, it will press the 'W' key to start the bus. What are the possible chunk types that you would recommend for this model?"

The generated response from ChatGPT4 suggested several chunk types, including "visual-cue", "state", "action", "bus-location", and "deviation," which would facilitate the model's perception, decision-making, and actions. However, Bard suggested chunk types such as "look around", "Start Bus", "Drive", and "Crash", which indicates that Bard misidentified the chunk types for the sequence of production rules.

Human In the Loop and Query Initial Rule

Based on the LLMs' responses for chunk types, we identified "state" and "visual-cue" as the two slots for the "drive" chunk, considering the task's requirements and to reduce computing load. Subsequently, we provided a second prompt containing an explanation of the chunk and slots that this model will use, and we queried LLMs for the first perception-related production, which involves using the visual module to look

around the environment.

The ChatGPT4 model first specifies the information that will be encoded for the model by defining chunk-types for the task. Then, it attempts to start the model by putting the chunk into the goal buffer, and its initial production rule generally follows the LHS-RHS format, which its RHS buffers and modules satisfying the conditions on the LHS of the production.

However, the code generated by Bard is completely wrong, as it doesn't show any trace of ACT-R model syntax. Consequently, we decided to continue with only ChatGPT4.

Test and Debug the Code The resulting ACT-R code from ChatGPT4 was then tested and debugged in the Emacs environment using the eLisp slime mode package. Following are corrections we made, showing the problem and the resolution.

1. Problem: Fail to define the model.
Resolution: Include the necessary ACT-R model declaration in the beginning by adding: "define-model Drive-Bus".
2. Problem: Absence of visual buffer check for initial state.
Resolution: Add the following to the RHS to empty the visual buffer for model initialization: "+visual-location :attended nil".
3. Problem: Misidentification of chunk type.
Resolution: Change "isa move-attention" to "cmd move-attention".
4. Problem: Missing visual buffer LHS check.
Resolution: Add the following to the LHS to meet the requirement of the visual buffer check: "?Visual >State free".
5. Problem: Miss slot argument.
Resolution: Add "Center-line" as the slot visual-cue argument when defining the "drive" chunk.

Human In the Loop and Query Following Rule We then prompted ChatGPT4 with the corrected code and queried it to generate the next production rule of "if it sees the center-line, press the "W" key to start the bus". Specifically, our human-in-the-loop intervention comprised of two cases. In the first case, we provided only the corrected code, while in the second case, we provided the corrected code along with a description of debugging. We then tested, compared, and analyzed the two generated code pieces from these two cases.

The two generated codes show differentiation in the quality. Specifically, in the prompts that contain corrected code and debugging reasons. These prompts generate syntax-correct production rule that meet the requirements of sequential firing and LHS-RHS matching. However, the prompts that contain only corrected code still produce one syntax error, where in the LHS side "=visual>" was mistakenly written as "?visual>", making the model unable to meet the requirements of production rule sequential firing. In addition, we have identified some general mistakes for ChatGPT4-generated ACT-R models that involve perceptual and motor behaviors, and we list the corresponding solutions below.

```
CL-USER> (run 1)
0.000 GOAL SET-BUFFER-CHUNK GOAL GOER NIL
0.000 VISION SET-BUFFER-CHUNK VISUAL-LOCATION CHUNK0 NIL
0.000 VISION visicon-update
0.000 PROCEDURAL CONFLICT-RESOLUTION
0.050 PROCEDURAL PRODUCTION-FIRED READY
0.050 PROCEDURAL CLEAR-BUFFER VISUAL-LOCATION
0.050 VISION Find-location
0.050 VISION SET-BUFFER-CHUNK VISUAL-LOCATION CHUNK0
0.050 PROCEDURAL CONFLICT-RESOLUTION
0.100 PROCEDURAL PRODUCTION-FIRED LOOK-AROUND
0.100 PROCEDURAL CLEAR-BUFFER VISUAL-LOCATION
0.100 PROCEDURAL CLEAR-BUFFER VISUAL
0.100 PROCEDURAL CONFLICT-RESOLUTION
0.185 VISION Encoding-complete CHUNK0-1 NIL
0.185 VISION SET-BUFFER-CHUNK VISUAL CHUNK1
0.185 PROCEDURAL CONFLICT-RESOLUTION
0.235 PROCEDURAL PRODUCTION-FIRED PRESS-W-KEY
Pressing W key
```

Figure 2: Output from the ACT-R model created with ChatGPT.

First, we noticed that the manual buffer lacked functionality. To rectify this issue, we manually added the appropriate manual command by checking the motor buffer in the LHS and implementing the correct functional keyboard command in the RHS. We did change "!eval! (output - key "w")", which has no functionality, to:

```
+manual>
ISA punch
hand right
finger index
```

Second, we observed that ChatGPT4 did not include Visicon features or install motor and keyboard devices, which are essential for the model's interaction and decision-making processes. To overcome this limitation, we added the necessary Visicon features and installed the required motor and keyboard devices to ensure a comprehensive cognitive simulation.

We recommend that the ACT-R models, which are built with the help of ChatGPT4 and using visual and manual modules, add the following lines of code to enable the modules to function properly.

```
(install-device '( "motor" "keyboard" ))
(add-visicon-features '(screen-x XX screen-y XX value center-line))
```

So, despite deep learning, the level of knowledge needed to create an interactive ACT-R model, ChatGPT4 needed some further, advanced knowledge.

Model Evaluation In Figure 2, we list the ACT-R model output that has been built using the help from ChatGPT4.

As Figure 2 shows, the prompts that we fed into ChatGPT 4 have generated a model that performs the corresponding behavior as requested. For the declarative memories, the model has the necessary chunk-types and slots to decide the driving state based on the visual cues it perceives. For the production rules, it sequentially fires get ready, then looks around, sees the visual pattern using the visicon, and then uses the manual buffer to press the key. This model has the potential to interact with the unmodified and novel simulation environment, and might be helpful to some modelers.

Soar Model

The simulation task involves the model being able to analyze student responses in a database and identify the dominant intelligence type. We succinctly present the procedure of prompting for Soar code using LLMs, emphasizing the differences from the procedure of the ACT-R model. We then use a table to list the main findings, including the prompt context, differentiation output provided by ChatGPT4, and Bard.

Provide Context and Prompt for Declarative Chunk(s)

The first prompt outlined the context and requirements of the simulation, asking for recommended chunk types for the model. The generated answer suggested several types of chunks, including “smart-cue”, “state”, “action”, “intelligence-location” and “student”, which would facilitate the perception, decision-making and actions of the model.

Human In the Loop, Query for Codes, Test and Debug

Based on the initial prompt response, we identify “state” and “time cue” as the two slots for the “student” chunk, simplifying the model. Later, we provide a second prompt to explain the types of chunks and slots this model will use, and ask ChatGPT4 and Bard to generate the code for Soar, which involves using the temporal buffer to examine the types of intelligences.

The resulting code was then tested and debugged in the Windows-10 environment using SoarDebugger 9.6.1, below we list common mistakes that ChatGPT4 and Bard made when creating the models:

Prompt Context 1: Introduction about Soar (intention to perform a pre-load in the tool) :

Chat-GPT: Wrote persuasive text, is this not “useful code”? appropriate to the request, and demonstrated great accuracy with regard to context in the response, however some details are not 100% correct.

Bard: Wrote persuasive text, suited to the request, and demonstrated great accuracy in relation to the context in the response, in this case the highlight is the ability to try to sound like a human being expressing concepts in a personal way.

Prompt Context 2: 1st Question on how to install and configure Soar on a PC with Windows10 :

Chat-GPT: Answer based on Soar’s official website.

Bard: Incorrect answer, persuasively quoted a non-existent tool: “Soar Toolkit” —possible delusion?

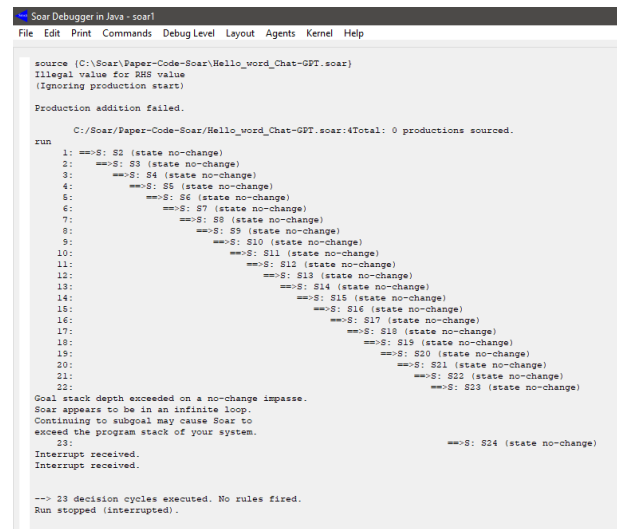
Prompt Context 3: 1st Intervention requesting further clarification on the procedure :

Chat-GPT: Almost everything right, but lacked steps to make it work.

Bard: Continued with a completely wrong answer.

Prompt Context 4: 2nd Question, considering the scenario of the educational study, to identify the type of dominant intelligence in a student :

Chat-GPT: It demonstrated an interesting capacity to approach the problem, but it had some confusion mainly in relation to syntax.



```
Soar Debugger in Java - soar!
File Edit Print Commands DebugLevel Layout Agents Kernel Help

source (C:\Soar\Paper-Code-Soar\Hello_word_Chat-GPT.soar)
Illegal value for RHS value
(Spawning production starts)

Production addition failed.

C:/Soar/Paper-Code-Soar/Hello_word_Chat-GPT.soar:(Total: 0 productions sourced.

run
1: ==>S: S2 (state no-change)
2: ==>S: S3 (state no-change)
3: ==>S: S4 (state no-change)
4: ==>S: S5 (state no-change)
5: ==>S: S6 (state no-change)
6: ==>S: S7 (state no-change)
7: ==>S: S8 (state no-change)
8: ==>S: S9 (state no-change)
9: ==>S: S10 (state no-change)
10: ==>S: S11 (state no-change)
11: ==>S: S12 (state no-change)
12: ==>S: S13 (state no-change)
13: ==>S: S14 (state no-change)
14: ==>S: S15 (state no-change)
15: ==>S: S16 (state no-change)
16: ==>S: S17 (state no-change)
17: ==>S: S18 (state no-change)
18: ==>S: S19 (state no-change)
19: ==>S: S20 (state no-change)
20: ==>S: S21 (state no-change)
21: ==>S: S22 (state no-change)
22: ==>S: S23 (state no-change)

Goal stack depth exceeded on a no-change impasse.
Soar appears to be in an infinite loop.
Continuing to subgoal may cause Soar to
exceed the program stack of your system.

23:
Interrupt received. ==>S: S24 (state no-change)
Interrupt received.

--> 23 decision cycles executed. No rules fired.
Run stopped (interrupted).
```

Figure 3: Output from the Soar model created with Chat-GPT

Bard: Didn’t use the basic standard Soar syntax structure (“If” “then”...) very wrong!

Prompt Context 5: 2st Intervention requesting further clarification on the procedure :

Chat-GPT: Made a good suggestion on how to run the model.

Bard: Although wrong, it appeared to deduce information in the context of computational systems and gave interesting but wrong data.

Prompt Context 6: 1st Order, requesting to Write a Hello World code :

Chat-GPT: Wrote almost correct code, missing only two details to work.

Bard: Wrote a code that was completely wrong in terms of syntax, but coherent in terms of intent.

Model Evaluation Figure 3 shows that the code generated by ChatGPT4 and added to Soar Debugger has been continuously rejected and ended up triggering the Debugger’s safe mode so as not to enter an infinite loop in the wrong way, it is easy to notice that few adjustments would be necessary for the code to run perfectly well.

Prompt Patterns that Maximize LLMs Interaction for Artificial Cognitive Architectures: A Framework for Evolving Conversational Excellence

This section presents the takeaway from our previous experiments in generating cognition models using LLMs. It addresses the demand for a framework that allows effective interactions, contextual understanding, and synchronous adaptation with evolving LLMs. Each prompt pattern is accompanied by implementation samples or examples of the prompt.

Initiation and Setting the Context using Persona

To initiate interaction, users must begin their input with a specific keyword or phrase denoting the domain or topic of interest. For example, if the user wants to discuss "Soar Architecture," they can start with an open question to initiate LLMs' train of thoughts and create an opportunity of synchronous training "Let's discuss Soar architecture, and what do you know about Soar architecture?" This also allows LLMs to establish context and adapt responses accordingly.

In addition, as new versions of ACT-R and Soar are released, users must explicitly tell the model which version is being used at the start of the conversation. This allows LLMs to adapt its responses in sync with model-specific improvements and capabilities. Moreover, users should using the persona "Next, I want you to act as Soar modeler" to increase LLMs' situational awareness for the tasks.

Multi-Turn Talk Within the Human-in-the-Loop Approach

Rather than relying on single-turn interactions (multiple requests within a single prompt), it is beneficial for the user to create a relationship based on multi-turn conversations (sequence of prompts). By maintaining context within the same channel, LLMs can "recall" previous exchanges, leading to coherent and contextualized responses. This approach is advantageous for developing an evolving and ongoing train of thought (Lieto 2021).

This sequential prompting approach integrates well with the human-in-the-loop approach, especially when the LLMs face constrained and limited situations. For example, when building the ACT-R model using ChatGPT4, we first create a multi-turn conversation by dividing the prompt to create the model into four sequential interactions. These talks include prompts for chunk types, the first production rule, debugging and testing, and sequential production rules. We also assist ChatGPT4 in evolving its thought process in building the model by synchronously training it with the reasons for picking up the chunk types it recommended and debugging the first round of code generation.

Synchronous Domain-Specific Training

Supplement LLMs training with domain-specific data and this enhances the model's understanding of the complexities and nuances of these architectures, leading to more informed and contextually appropriate responses (Baraka, Alves-Oliveira, and Ribeiro 2020).

We emphasize that the training should at least occur in two timestamps. First, it should happen during the initiation of the context when users ask LLMs what they know about the domain context. Users should provide feedback, additional information, or corrections during this first round of talk to help LLMs set up the context. Second, the training should happen in the iterative code-building process by providing the LLMs with the debugging reasons. We found out that within the synchronous training during the initial debugging of the code, ChatGPT4 generates higher quality of

sequential code compared to data without debugging training.

Provide Diversified Meta-Communications

Periodic meta-communications can further improve LLMs model performance over time (Silva, Sá, and Lima Junior 2019). We recommend at least two types of meta-communications and encourage diversified adoption. First, users can ask LLMs clarifying questions to seek better understanding of user intent and provide more accurate answers. Second, users can provide LLMs with feedback on the output generated. For example, they can say, "This answer was helpful" or "Could you elaborate on this more?"

A GitHub repository (Souza et al. 2023) has been established to host the prompt patterns framework and encourage contributions. This collaborative platform invites researchers, developers, and enthusiasts to actively participate, share insights, and collectively improve the interactivity between LLMs and artificial cognitive systems.

Conclusion and Discussion

Our study contributes to the emerging area of combining language models and cognitive modeling principles. We present a systematic and detailed design and development process of two independent models based on ACT-R and Soar using LLMs as interactive interfaces. In addition, we identify and address common challenges and errors that arise when using LLMs in modeling with ACT-R and Soar, and provide a set of solutions. It is worthy to note that the initial code was not correct enough to run on its own. This can be contrasted with success stories told about working Java and Visual Basic code, where existing programs may be used. However, in this case, the semantics of these languages are more complicated, and there may not be enough worked examples that were used to create these LLMs.

To facilitate future research in this domain, we propose a rapid framework that other researchers can adopt when employing LLMs for the development of cognitive agents for the Soar Architecture or ACT-R model. This framework will serve as a valuable resource for those interested in harnessing the potential of immediate engineering to enhance their cognitive simulations.

The proposed framework lays the groundwork for improved interaction between users and LLMs, particularly in the context of artificial cognitive architectures like ACT-R and Soar. By prioritizing context setting, multi-turn conversations, and user feedback, the framework fosters an environment where LLMs features can be optimally leveraged. Additionally, incorporating domain-specific training and fine-tuning ensures that LLMs remains current and aligned with the evolving demands of users and the field of Artificial Cognitive Architectures. With these steps, the interaction becomes a mutually beneficial process, driving the advancement of LLMs and cognitive architecture modeling. Researchers can readily adopt this framework to explore the potential of LLMs in enhancing cognitive simulations.

In conclusion, this case study serves as a pioneering effort in exploring the synergies between ChatGPT4, Google Bard,

Soar, and ACT-R. It shows the promise of immediate engineering using LLMs for more accessible and accurate cognitive model development. By presenting an attractive application in the simulated driving task domain, we anticipate that this work will inspire further research and development in the integration of language models with cognitive modeling frameworks.

References

- Anderson, J. R. 2009. *How can the human mind occur in the physical universe?* Oxford University Press.
- Baraka, K.; Alves-Oliveira, P.; and Ribeiro, T. 2020. *An Extended Framework for Characterizing Social Robots*, volume 12, 21–64.
- Bothell, D. 2017. ACT-R 7 Reference Manual. Available at actr.psy.cmu.edu/wordpress/wpcontent/themes/ACT-R/actr7/reference-manual.pdf. Accessed February 2023.
- Cerf, V. G. 2023. Large Language Models. *Commun. ACM*, 66(8): 7.
- Gardner, H. 1993. *Multiple intelligences: The theory in practice*. Basic Books.
- GitHub. 2023. <https://github.com/f/awesome-chatgpt-prompts>, GitHub - f/awesome-chatgpt-prompts — github.com, Accessed: 2023-07-17.
- Google. 2023. Google Bard. Retrieved from <https://www.google.com/bard> [Online; accessed July 17-2023].
- Huizinga, M.; Baeyens, D.; and Burack, J. A. 2018. Executive function and education. *Frontiers in Psychology*, 9: 1357.
- Laird, J. E. 2019. *The Soar cognitive architecture*. MIT press.
- Lieto, A. 2021. *Cognitive design for artificial minds*. ISBN 9781315460536.
- Newell, A. 1990. *Unified Theories of Cognition*. MA:Harvard University Press.
- OpenAI. 2023. ChatGPT: Large-Scale Generative Language Models for Automated Content Creation, <https://openai.com/blog/chatgpt/>, 2023, [Online; accessed July 17-July-2023].
- Pew, R. W.; and Mavor, A. S. 1998. *Modeling human and organizational behavior-application to military simulations: Applications to military simulations*. Washington, DC: National Academy Press.
- Ritter, F.; Tehranchi, F.; and Oury, J. 2019. ACT-R: A cognitive architecture for modeling cognition. *Wiley Interdisciplinary Reviews: Cognitive Science*, 10: e1488.
- Ritter, F. E.; Kukreja, U.; and Amant, R. S. 2007. Including a model of visual processing with a cognitive architecture to model a simple teleoperation task. *Journal of Cognitive Engineering and Decision Making*, 1(2): 121–147.
- Schwartz, D.; Tehranchi, F.; and Ritter, F. E. 2020. Drive the bus: Extending JSegMan to drive a virtual long-range bus. *Proceedings of ICCM, The 18th International Conference on Cognitive Modeling*, 241-246.
- Shakir, A. 2002. Assessment of Models of Human Decision-Making for Air Combat Analysis. [Unpublished technical report. DERA Farnborough. Abstract put on the web with permission, at <http://acs.ist.psu.edu/papers/shakir02-abstract.pdf>].
- Silva, J.; Sá, N.; and Lima Junior, W. 2019. Licklider’s fundamental ideas in the ”man-computer symbiosis” reemerge in the concepts of cognitive computing: a bibliometric study. Proc. of the XI International Brazilian Meeting in Cognitive Science. At: São Paulo.
- Souza, R. F.; Wu, S.; Ritter, F. E.; and Lima Jr, W. T. 2023. https://github.com/rodrigo-ferreira-unifesp/LLMs_interaction_for_artificial_cognitive_systems — <https://ury1.com/lmms-act-r-soar> —. Accessed 12-July-2023.
- Tehranchi, F.; Bagherzadeh, A.; and Ritter, F. E. 2023. A user model to directly compare two unmodified interfaces: A study of including errors and error corrections in a cognitive user model. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*. In press.
- White, J.; Fu, Q.; Hays, S.; Sandborn, M.; Olea, C.; Gilbert, H.; and Schmidt, D. C. 2023. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. ArXiv preprint arXiv:2302.11382.
- Wu, S.; Bagherzadeh, A.; Ritter, F.; and Tehranchi, F. 2023. Long road ahead: Lessons learned from the (soon to be) longest running cognitive model. *21st International Conference on Cognitive Modeling (ICCM) at the University of Amsterdam, the Netherlands*. In press.