2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)

CodeACT-R: A Cognitive Simulation Framework for Human Attention in Code Reading

Yueke Zhang[1], Zihan Fang[1], Greg Trafton [2], Daniel Levin[1], Kevin Leach[1], and Yu Huang[1],
[1]*Vanderbilt University, Nashville, USA*
{yueke.zhang, zihan.fang, daniel.t.levin, kevin.leach, yu.huang}@vanderbilt.edu
[2]*US Naval Research Laboratory, Washington, D.C., USA*
greg.j.trafton.civ@us.navy.mil

***Abstract*—Reading code is a fundamental activity in both software engineering and computer science education. Understanding the cognitive processes involved in reading code is crucial for identifying effective cognitive strategies, which can inform teaching methods and tooling support for developers. However, collecting large human subject eye tracking datasets, especially for programming tasks, is often costly and time-consuming, limiting its scalability and applicability. To address this issue, we present CodeACT-R, the first cognitive simulation framework tailored for code reading, based on the well-established Adaptive Control of Thought—Rational (ACT-R) architecture from cognitive science. CodeACT-R simulates how humans read code and requires only a small, manageable amount of human data to initiate the simulator design, offering a cost-effective and scalable alternative to traditional data collection methods like eye tracking.**

Specifically, we first collected real human visual attention data from 48 programmers reading code using eye tracking. These data were then used to develop CodeACT-R, enabling the simulation of human-like code reading behaviors. Our evaluation demonstrates that CodeACT-R is capable of simulating visual attention patterns (i.e., scanpaths) that closely resemble real-world human attention patterns, also accounting for up to 87% of observed pattern variations.

***Index Terms*—Cognitive Simulation, Eye Tracking, Code Reading, Program Comprehension**

I. Introduction

Code reading is a cornerstone of software engineering, enabling developers to understand, maintain, and debug complex systems [1], [2]. The resulting visual *scanpaths*—representing the sequence of locations where a user's gaze pauses (i.e., fixations)—reveal how developers process information during comprehension tasks [3], [4]. Recent advances have demonstrated that this human attention data can be leveraged to train significantly more effective AI models for downstream software engineering tasks, such as code summarization and comprehension difficulty prediction [5], [6]. These models learn from datasets pairing code snippets with their corresponding human visual attention patterns, marking a new direction for cognitively-aware AI.

However, the potential of these attention-guided models is constrained by a fundamental challenge: data scarcity. Deep learning models realize their full potential only when trained on vast amounts of data [7], [8], but gathering large-scale human eye-tracking data is difficult. Human studies are expensive, time-consuming, and hard to scale, typically yielding data from only small cohorts of programmers [9]–[12]. This data bottleneck forces researchers into suboptimal compromises. Current approaches either rely on synthetic data augmentation techniques from computer vision that produce unrealistic and homogeneous attention patterns [5], [13], [14], or are confined to simplified tasks where limited real data is sufficient [6], thus failing to address more complex, real-world problems.

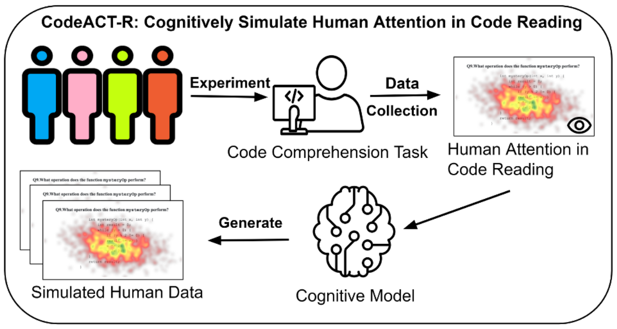


Fig. 1. The development pipeline of CodeACT-R: It begins with collecting scanpaths from programmers, then uses these scanpaths to construct a cognitive simulation model based on the ACT-R architecture, which can generate cognitively plausible, human-like scanpaths for a given code snippet.

To overcome this data bottleneck, we turn to cognitive simulation. We leverage the *Adaptive Control of Thought—Rational (ACT-R)* architecture, a prominent computational framework from cognitive psychology designed to simulate human cognitive processes with high fidelity [15], [16]. ACT-R has been successfully applied to model human visual attention in diverse and complex tasks, including menu selection and driving [17], [18], demonstrating its robustness and reliability for generating plausible human-like behavior.

Inspired by this work, we present *CodeACT-R*, a novel cognitive simulation framework tailored for the code reading process. As illustrated in Figure 1, our approach begins by collecting a small set of human scanpaths. This data is then used to construct a cognitive model within the ACT-R architecture, which can generate a large volume of plausible, human-like scanpaths for a given code snippet. Our evaluation shows that scanpaths from CodeACT-R are 29% closer to real human scanpaths than baseline methods and cover 33% more real-world patterns.

The contributions of this paper are:

1) We present CodeACT-R, the first cognitive simulation framework to generate human-like scanpaths for code reading.
2) We demonstrate the effectiveness of CodeACT-R through an evaluation showing that its generated scanpaths significantly outperform baselines—achieving a 29% improvement in similarity to real data even with no prior examples—and can capture up to 87% of real-world attention patterns when minimally seeded.

We share the result and implementation code of this study https://github.com/zyueek/CodeACT-R

II. Related Work

Though new to SE community, the *ACT-R (Adaptive Control of Thought—Rational)* architecture is a well-established architecture in cognitive psychology that provides a theoretical and computational framework for understanding and simulating human cognitive processes [15], [16], [19]. Researchers use ACT-R to understand the low level process that people engage in to perform particular tasks. This process is a computational model that can be used to simulate, validate, and analyze different tasks, participant populations, AI models, and cognitive processes [16], [18], [20], [21]. ACT-R has been successfully and widely applied to simulate human visual attention in various tasks, such as Nilsen's random menu selection task [22], Byrne and Anderson's visual search tasks [17], and Salvucci and Gray's driving tasks [18], and proved to be versatile, robust and reliable [23]–[25].

Recent research has demonstrated that integrating human behavioral and cognitive signals can enhance the performance of deep learning models for software engineering tasks [26]. For instance, recent studies have successfully used eye-tracking data (gaze) to improve neural code summarization, showing that models guided by human attention generate more relevant and accurate summaries [4], [5]. Similarly, researchers have leveraged gaze patterns to predict code comprehension difficulty and have even used functional magnetic resonance imaging (fMRI) data to align neural models of code with brain activity during program comprehension [6], [27]. These efforts validate the premise that human data is a valuable resource for building better AI tools.

III. Design of CodeACT-R

We designed CodeACT-R to simulate scanpaths at the statement level. The design involves collecting human scanpaths, constructing a declarative memory, and defining production rules to generate new, simulated scanpaths.

A. Data Collection and Scenario Description

We recruited 48 participants from a computer science course and used a 60hz Tobii Pro Fusion eye-tracker to collect their scanpaths on 11 C++ code comprehension tasks. The raw gaze data was processed to generate scanpaths, which are sequences of attended statement numbers. We designed CodeACT-R to operate in two scenarios:

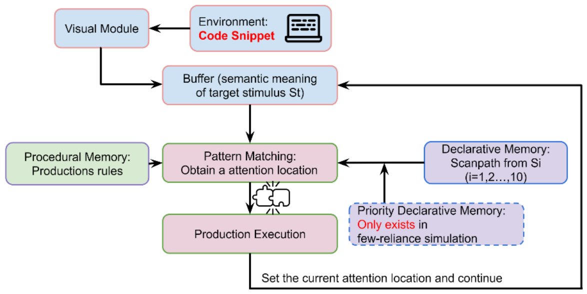
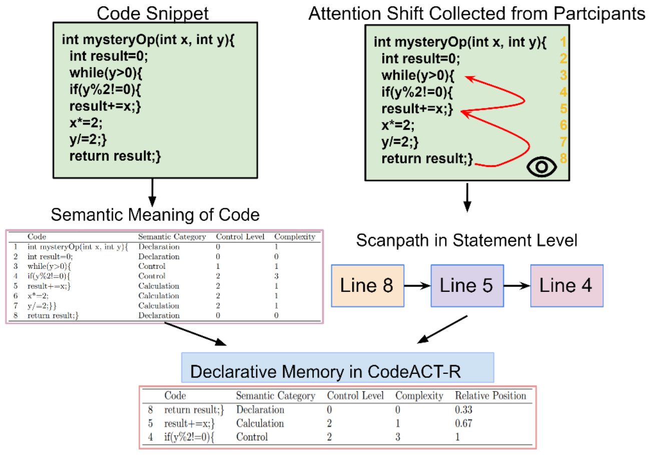

Visual Module
Environment: Code Snippet
Buffer (semantic meaning of target stimulus St)
Procedural Memory: Productions rules
Pattern Matching: Obtain a attention location
Declarative Memory: Scanpath from Si (i=1,2...,10)
Priority Declarative Memory: Only exists in few-reliance simulation
Production Execution
Set the current attention location and continue

Fig. 2. The architecture of the CodeACT-R in our study.


Code Snippet
Attention Shift Collected from Partcipants
Semantic Meaning of Code
Scanpath in Statement Level
Line 8
Line 5
Line 4
Declarative Memory in CodeACT-R

```
int mysteryOp(int x, int y){
  int result=0;
  while(y>0){
  if(y%2!=0){
  result+=x;}
  x*=2;
  y/=2;}
  return result;}
```

	Code	Semantic Category	Control Level	Complexity
1	int mysteryOp(int x, int y){	Declaration	0	1
2	int result=0;	Declaration	0	0
3	while(y>0){	Control	1	1
4	if(y%2!=0){	Control	2	3
5	result+=x;}	Calculation	2	1
6	x*=2;	Calculation	2	1
7	y/=2;}}	Calculation	2	1
8	return result;}	Declaration	0	0

	Code	Semantic Category	Control Level	Complexity	Relative Position
8	return result;}	Declaration	0	0	0.33
5	result+=x;}	Calculation	2	1	0.67
4	if(y%2!=0){	Control	2	3	1

Fig. 3. An example of encoding code semantic meaning and human scanpath into declarative memory.

- **Zero-reliance:** Simulating scanpaths for a new code snippet for which no human data exists. The simulation relies only on data from other, different code snippets.
- **Few-reliance:** Simulating additional scanpaths for a code snippet where a few human scanpaths are available.

B. The Components of CodeACT-R

In Figure 2, we present the CodeACT-R architecture used in our study. The components of the architecture include:

- Buffer: The CodeACT-R encodes the target code snippet S_t from the environment and captures the semantic meaning, which is then stored in the ACT-R buffer.
- Declarative Memory: Declarative memory contains code semantic information and corresponding scanpath from $S_i, i = 1, 2, ...10$. For the few-reliance scenario, we use existing scanpaths for the target stimulus S_t to build a priority declarative memory, enhancing the use of scanpath data.
- Production Rule: Production rules perform pattern matching between the declarative memory and the buffer.
- Pattern Matching: The pattern matching process employs production rules from procedural memory to match patterns between the buffer and declarative memory, and then executes these rules to determine attention locations in the scanpath.

C. Constructing the Declarative Memory from Human Scanpaths

The core of CodeACT-R is its declarative memory, which stores structured knowledge extracted from the collected human scanpaths. As illustrated in Figure 3, populating this memory involves encoding each fixated statement from a human scanpath along two dimensions: its semantic properties and its temporal position within the reading sequence.

First, to capture the semantics of the code, we analyze each statement using an Abstract Syntax Tree (AST) and encode it with three features:

- **Semantic Category (Sem):** Classifies the statement's purpose, such as a control flow statement ('if', 'for'), a calculation, or a variable declaration.
- **Control Level (Con):** Measures the nesting depth of the statement within control structures.
- **Complexity (Comp):** Measures the statement's syntactic complexity based on the number of operators and function calls.

Second, to capture the temporal dynamics of code comprehension, we compute and store the relative position of each fixation within its scanpath. This is a normalized value indicating how far into the reading process a statement was viewed. For instance, in a 10-step scanpath, the first fixated statement is at a relative position of 10%, while the fifth is at 50%. This encoding allows the model to learn the general flow of comprehension—for example, which types of statements are typically examined at the beginning, middle, or end of a reading session.

In the *few-reliance* scenario, where a small number of human scanpaths for the target code snippet are already available, these specific scanpaths are encoded using the same method but are stored in a separate, high-priority declarative memory to guide the simulation more directly.

D. The Scanpath Simulation

For simulating scanpath using CodeACT-R. First, we set the scanpath length, and then conduct pattern-matching using production rules.

1) Estimating Scanpath Length: The model first estimates an appropriate length for the simulated scanpath. This estimation is guided by the principle that more complex code requires longer reading times. The model calculates the *Complexity* of the target code snippet based on its statements' features. It then compares this complexity value to the complexities of the code snippets associated with the human scanpaths stored in its declarative memory. Based on this comparison, it selects a probable scanpath length.

2) Different Pattern Matching Approaches: Once a length is determined, the model generates the scanpath one fixation at a time. It uses production rules to retrieve information from its declarative memory and select the next statement to fixate on in the target code. We implemented three distinct pattern-matching strategies to model different cognitive reading behaviors.

- **Semantic Pattern Matching:** This is the default strategy. For each attention step, it finds a statement in declarative memory with a similar relative position. It then selects a statement from the target code that matches the semantic features (Sem, Con) of the retrieved memory item. If multiple statements match, the one with the highest complexity is chosen.
- **Quick Pattern Matching:** Used for short scanpaths (length < 4), this strategy models heuristic reading. It either focuses on the most complex statements or traces variables backward from the return statement, mimicking expert shortcuts [28].
- **Cache Pattern Matching:** Used in the few-reliance scenario. This strategy analyzes the transition probabilities between statement types (e.g., a 'declaration' is often followed by a 'calculation') in the existing scanpaths for the target code. It then uses these learned probabilities to guide the simulation, prioritizing patterns already observed in that specific code.

For the zero-reliance scenario, CodeACT-R uses Quick Pattern Matching for short scanpaths and Semantic Pattern Matching otherwise. For the few-reliance scenario, it uses Cache Pattern Matching for longer scanpaths to leverage the available data.

IV. Experiment

We conducted experiments to evaluate the accuracy and diversity of the scanpaths generated by CodeACT-R.

A. Metrics

We used two primary metrics for evaluation:

- **Average Value-Sensitive Levenshtein Distance (AVSL):** We introduce AVSL, an adaptation of the Levenshtein distance, to measure the similarity between a simulated and a real scanpath. Unlike the standard distance, AVSL's substitution cost is proportional to the difference between line numbers (e.g., mistaking line 2 for line 3 is a smaller error than mistaking it for line 10). A lower AVSL indicates higher similarity.
- **Coverage:** The percentage of unique real-world scanpaths that are "covered" by the simulation, meaning at least one simulated scanpath is similar to it (AVSL < 1). This metric assesses the diversity of the generated data.

B. Baselines and Evaluation Protocol

We compared CodeACT-R against two baselines: *Declarative Memory Scanpath (DMS)*, a random scanpath whose length is sampled from the entire dataset, and *Value Comparison Scanpath (VCS)*, a random scanpath whose length is sampled from the real scanpaths of the target stimulus. For each experimental condition, we generated 300 simulated scanpaths per code snippet to ensure a robust evaluation, as reflected in our result.

For the zero-reliance scenario, we used leave-one-out cross-validation (LOOCV), training on 10 stimuli and testing on the remaining one. For the few-reliance scenario, we used a

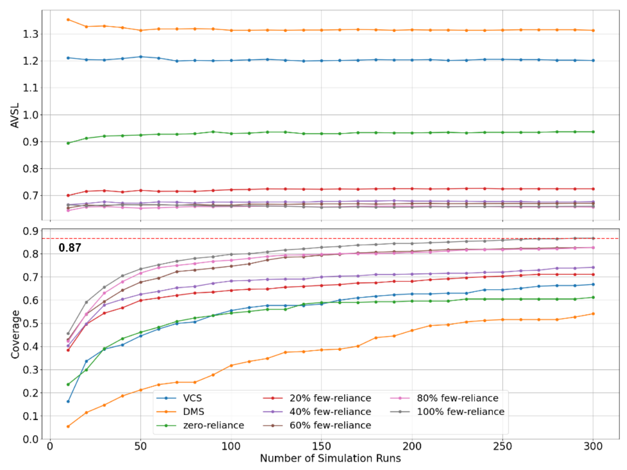


Fig. 4. AVSL (top) and Coverage (bottom) for CodeACT-R and baselines. Lower AVSL is better; higher Coverage is better. CodeACT-R consistently outperforms the baselines. The 100% few-reliance simulation achieves the best performance, covering up to 87% of real human scanpaths.

portion of a stimulus's real scanpaths (20% to 100%) to build the priority memory and validated against the rest.

V. Results

This section presents the results of our experimental evaluation, following the protocol described previously. We analyze the performance of CodeACT-R in terms of similarity (AVSL) and diversity (Coverage) across our two primary scenarios.

A. *Zero-Reliance Scenario Performance*

We assessed the extent to which CodeACT-R can generate cognitively plausible scanpaths without any prior human data for a given code snippet. The results show that our approach outperforms established baselines. As illustrated in Figure 4 (top), the zero-reliance simulation achieved a final Average Value-Sensitive Levenshtein Distance (AVSL) of 0.93. This score indicates that the generated scanpaths are substantially more similar to real human data than those from the DMS (AVSL=1.31) and VCS (AVSL=1.2) baselines, representing a 29% improvement in similarity over DMS. Regarding diversity, the zero-reliance simulation successfully covered approximately 60% of unique real-world scanpath patterns (Figure 3, bottom), demonstrating the ability to generate a wide and plausible range of attention patterns even from scratch.

B. *Few-Reliance Scenario Performance*

We evaluated how effectively CodeACT-R leverages a small amount of existing human data to enhance its simulations. The findings confirm that even a minimal seed of real data yields performance gains. As shown in Figure 4, incorporating just 20% of a stimulus's real scanpaths dramatically lowers the AVSL to approximately 0.7, making the simulations closer to the validation data. As the volume of reliance data increases, both similarity (lower AVSL) and coverage (higher percentage) consistently improve. For instance, the 20% few-reliance model already achieves over 75% coverage, surpassing the baselines and the zero-reliance model by a wide margin. At its peak, the 100% few-reliance simulation achieves the best overall performance, covering 87% of observed human patterns and validating our framework's ability to generate an accurate and comprehensive set of simulated attention data.

In summary, the more few-reliance percentage also leads to an increase in AVSL and Coverage. For the coverage evaluation, the simulation can be saturated with 200 simulation runs. For the simulation with 60% and more few reliance, the coverage can reach up to 85%.

VI. Implication and Future Work

This study is the first to simulate human scanpaths in code reading using a cognitive model. Our framework, CodeACT-R, provides a scalable and cost-effective method for generating large, high-quality datasets of human-like attention data, which can help overcome the data acquisition bottleneck in software engineering research.

This work has two key implications. **First**, it encourages the adaptation of cognitive models to other software engineering tasks, providing insights into expert cognitive processes that can inform computer science education. **Second**, the simulated data can facilitate Human-AI training, enabling the development of more powerful and cognitively-aware AI tools for code generation, error prediction, and personalized learning.

The primary application of our large-scale simulated data is to enhance the performance and cognitive alignment of deep learning models, particularly Large Language Models (LLMs). Our future work will proceed in two main phases. **First**, we will leverage the validated CodeACT-R framework to generate a large-scale dataset of simulated human attention patterns. This dataset will encompass a diverse range of code snippets, programming languages, and task complexities, creating a rich, cognitively-grounded resource that is currently unavailable to the research community. **Second**, we will use this unique dataset to fine-tune or pre-train state-of-the-art LLMs for downstream software engineering tasks. We hypothesize that by training models on data reflecting human problem-solving strategies, we can instill a valuable inductive bias. This should lead to models that not only demonstrate superior performance in tasks like neural code summarization, automated bug detection, and code review, but also exhibit greater interpretability for LLM models.

VII. Conclusion

We conducted the first study to cognitively simulate human attention shifts in code comprehension tasks. We developed the CodeACT-R, based on the cognitive framework ACT-R. This model allows for two tasks: (1) zero-reliance, which simulates scanpaths even without existing data for a given stimulus, and (2) few-reliance, which generates additional data based on existing scanpaths for a stimulus. We evaluated how closely our simulated scanpaths align with real-world scanpaths and the diversity of attention patterns they capture. Our best CodeACT-R simulation covered up to 87% of unique real-world scanpaths.

References

[1] T. Busjahn and C. Schulte, "The use of code reading in teaching programming," in *Proceedings of the 13th Koli Calling international conference on computing education research*, 2013, pp. 3–11.

[2] T. Busjahn, C. Schulte, and A. Busjahn, "Analysis of code reading to gain more insight in program comprehension," in *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, 2011, pp. 1–9.

[3] K. Holmqvist, M. Nyström, R. Andersson, R. Dewhurst, H. Jarodzka, and J. Van de Weijer, *Eye tracking: A comprehensive guide to methods and measures*. oup Oxford, 2011.

[4] A. Bansal, C.-Y. Su, Z. Karas, Y. Zhang, Y. Huang, T. J.-J. Li, and C. McMillan, "Modeling programmer attention as scanpath prediction," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 1732–1736.

[5] Y. Zhang, J. Li, Z. Karas, A. Bansal, T. J.-J. Li, C. McMillan, K. Leach, and Y. Huang, "Eyetrans: Merging human and machine attention for neural code summarization," *arXiv preprint arXiv:2402.14096*, 2024.

[6] T. Alakmeh, D. Reich, L. Jäger, and T. Fritz, "Predicting code comprehension: A novel approach to align human gaze with code using deep neural networks," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 1982–2004, 2024.

[7] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

[8] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "Unixcoder: Unified cross-modal pre-training for code representation," *arXiv preprint arXiv:2203.03850*, 2022.

[9] Z. Sharafi, B. Sharif, Y.-G. Guéhéneuc, A. Begel, R. Bednarik, and M. Crosby, "A practical guide on conducting eye tracking studies in software engineering," *Empirical Software Engineering*, vol. 25, pp. 3128–3174, 2020.

[10] Z. Sharafi, Z. Soh, and Y.-G. Guéhéneuc, "A systematic literature review on the usage of eye-tracking in software engineering," *Inf. Softw. Technol.*, vol. 67, pp. 79–107, 2015.

[11] Y. Huang, K. Leach, Z. Sharafi, N. McKay, T. Santander, and W. Weimer, "Biases and differences in code review using medical imaging and eye-tracking: genders, humans, and machines," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 456–468.

[12] Z. Karas, A. Bansal, Y. Zhang, T. Li, C. McMillan, and Y. Huang, "A tale of two comprehensions? analyzing student programmer attention during code summarization," *ACM Transactions on Software Engineering and Methodology*, 2024.

[13] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 07, 2020, pp. 13 001–13 008.

[14] A. Mikołajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," in *2018 international interdisciplinary PhD workshop (IIPhDW)*. IEEE, 2018, pp. 117–122.

[15] J. R. Anderson, M. Matessa, and C. Lebiere, "Act-r: A theory of higher level cognition and its relation to visual attention," *Human–Computer Interaction*, vol. 12, no. 4, pp. 439–462, 1997.

[16] F. E. Ritter, F. Tehranchi, and J. D. Oury, "Act-r: A cognitive architecture for modeling cognition," *Wiley Interdisciplinary Reviews: Cognitive Science*, vol. 10, no. 3, p. e1488, 2019.

[17] M. D. Byrne, "Act-r/pm and menu selection: Applying a cognitive architecture to hci," *International Journal of Human-Computer Studies*, vol. 55, no. 1, pp. 41–84, 2001.

[18] D. D. Salvucci, "Modeling driver behavior in a cognitive architecture," *Human factors*, vol. 48, no. 2, pp. 362–380, 2006.

[19] J. R. Anderson, "Act: A simple theory of complex cognition." *American psychologist*, vol. 51, no. 4, p. 355, 1996.

[20] H. J. A. Fuller, M. Reed, and Y. Liu, "Integration of physical and cognitive human models to simulate driving with a secondary in-vehicle task," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, pp. 967–972, 2012.

[21] T. Schürmann and P. Beckerle, "Personalizing human-agent interaction through cognitive models," *Frontiers in Psychology*, vol. 11, p. 561510, 2020.

[22] A. J. Hornof and D. E. Kieras, "Cognitive modeling reveals menu search in both random and systematic," in *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, 1997, pp. 107–114.

[23] C. Lebiere, "The dynamics of cognition: An act-r model of cognitive arithmetic," *Kognitionswissenschaft*, vol. 8, no. 1, pp. 5–19, 1999.

[24] J. R. Anderson and L. J. Schooler, "Reflections of the environment in memory," *Psychological science*, vol. 2, no. 6, pp. 396–408, 1991.

[25] H. Chen, S. Liu, L. Pang, X. Wanyan, and Y. Fang, "Developing an improved act-r model for pilot situation awareness measurement," *IEEE Access*, vol. 9, pp. 122 113–122 124, 2021.

[26] C. Régis, J.-L. Denis, M. L. Axente, and A. Kishimoto, *Human-centered AI: A Multidisciplinary Perspective for Policy-makers, Auditors, and Users*. CRC Press, 2024.

[27] T. O. A. Binhuraib, G. Tuckute, and N. Blauch, "Topoformer: brain-like topographic organization in transformer language models through spatial querying and reweighting," in *ICLR 2024 Workshop on Representational Alignment*, 2024.

[28] J. A. Silva Da Costa and R. Gheyi, "Evaluating the code comprehension of novices with eye tracking," in *Proceedings of the XXII Brazilian Symposium on Software Quality*, 2023, pp. 332–341.