International Journal of Multidisciplinary Research and Growth Evaluation.

Cognitive Architecture for Adaptive Problem-Solving and Computational Models of Expert Knowledge Acquisition in Computer Science Education

Pamba Shatson Fasco
Department of Computer Science, School of Mathematics and Computing, Kampala International University, Uganda

* Corresponding Author: **Pamba Shatson Fasco**

Article Info
ISSN (online): 2582-7138
Volume: 06
Issue: 03
May-June 2025
Received: 10-04-2025
Accepted: 08-05-2025
Page No: 948-959

Abstract
The researcher presents a comprehensive investigation into cognitive architectures for adaptive problem-solving with specific application to computer science education. This study develops and evaluates a computational model that simulates the acquisition of expert knowledge in programming and algorithmic reasoning. Through rigorous experimentation and analysis, the researcher identifies key mechanisms that facilitate the transformation of cognitive structures during skill development. The results demonstrate significant advancements in understanding how novice programmers transition to expert status, revealing distinct cognitive patterns that emerge during this progression. These patterns include the formation of specialized mental schemas, the development of chunking mechanisms for efficient information processing, and the emergence of sophisticated heuristic strategies. The findings contribute to both theoretical understanding of expert cognition and practical applications in computer science pedagogy, offering evidence-based approaches for designing educational interventions that accelerate expertise development.

Keywords: cognitive architecture, adaptive problem-solving, expert knowledge, computational modeling, computer science education, skill acquisition, learning trajectories

1. Introduction

Computer science education faces persistent challenges in developing effective pedagogical approaches that facilitate the transition from novice to expert programmer. Despite significant advances in educational technology and teaching methodologies, many students continue to struggle with core programming concepts and problem-solving strategies. As Anderson *et al*. (2019) [19] observe, the acquisition of programming expertise involves complex cognitive processes that remain inadequately understood and modeled. This research gap hampers the development of educational interventions that could accelerate expertise development in computer science domains.

The researcher posits that computational models of cognition offer a promising approach to understanding and addressing these challenges. Current educational theories explain broad patterns of learning but lack the precision needed to inform targeted interventions in complex domains like computer science. Computational cognitive architectures, by contrast, can capture the fine-grained mechanisms underlying expertise development. These architectures can simulate how knowledge structures evolve during the progression from novice to expert status, providing insights that purely descriptive theories cannot offer.

A critical gap in the existing literature concerns the transformation of knowledge structures during programming skill acquisition. While research has documented differences between novice and expert programmers (Robins *et al*., 2023) [24], the cognitive processes that facilitate this transition remain underspecified. How do fragmentary conceptual understandings coalesce into sophisticated mental models? What mechanisms enable the development of pattern recognition capabilities that characterize expert problem-solving? These questions require detailed computational models that can simulate cognitive changes at multiple levels of abstraction.

This study aims to develop a cognitive architecture that accurately models the transition from novice to expert problem-solving in computer science domains. The architecture integrates mechanisms for knowledge representation, pattern recognition, and strategic planning—key components of programming expertise. By implementing and validating this architecture, the researcher seeks to advance both theoretical understanding of expertise development and practical approaches to computer science education.

The significance of this work lies in its potential to transform educational approaches through computational modeling of learning trajectories. Traditional educational designs often rely on intuitive notions of skill progression that may not align with actual cognitive development patterns. Grover and Pea (2018) [12] argue that evidence-based computational models can provide more reliable guidance for curriculum design, identifying optimal sequences of concept introduction and practice opportunities. Furthermore, these models can inform the development of intelligent tutoring systems that adapt to individual learning patterns, potentially accelerating expertise development.

This dissertation is organized into seven chapters. Following this introduction, Chapter 2 reviews relevant literature on cognitive architectures, expertise development, and computer science education. Chapter 3 presents the theoretical framework guiding the research, while Chapter 4 details the methodology for model development and validation. Chapter 5 reports the results of computational experiments testing the architecture's ability to simulate expertise development. Chapter 6 discusses theoretical and practical implications of these findings, and Chapter 7 concludes with a summary of contributions and directions for future research.

2. Literature Review

2.1 Cognitive Architectures in AI

The development of cognitive architectures represents a significant endeavor to model human cognition computationally. The researcher traces this development from early symbolic systems of the 1970s and 1980s to contemporary hybrid approaches that integrate multiple processing mechanisms. Early cognitive architectures such as GPS (General Problem Solver) established fundamental principles of problem-solving through means-ends analysis, while later systems incorporated more sophisticated mechanisms for knowledge representation and learning (Laird *et al*., 2017) [19].

ACT-R (Adaptive Control of Thought-Rational) represents one of the most influential cognitive architectures, offering a comprehensive framework for modeling cognitive processes. Anderson (2007) [1] describes how ACT-R integrates declarative and procedural knowledge modules with perceptual-motor interfaces, providing a unified theory of cognition that has been applied to numerous domains, including programming education. The architecture's distinction between declarative and procedural knowledge forms a theoretical basis for understanding expertise development, though its implementation of learning mechanisms remains somewhat constrained by predefined production rules.

SOAR provides an alternative architecture focused on problem-solving and learning through chunking operations. While SOAR excels at modeling goal-directed behavior and knowledge integration, its emphasis on symbolic processing limits its ability to capture the graded, probabilistic nature of human concept formation. Recent extensions to SOAR have attempted to address these limitations through reinforcement learning mechanisms, but challenges remain in modeling the flexibility characteristic of human expertise.

More recently, deep learning architectures have demonstrated impressive capabilities in pattern recognition and knowledge representation. However, as Wang and Patel (2019) [28] observe, these architectures often lack the interpretability and explicit knowledge structures that characterize human problem-solving. While they excel at extracting patterns from large datasets, they typically fail to capture the hierarchical, modular organization of human knowledge, which enables flexible adaptation to novel problems.

The limitations of existing approaches become particularly evident when modeling the flexibility and adaptability characteristic of human cognition. Current architectures struggle to capture how experts seamlessly move between different levels of abstraction, combine multiple knowledge sources, and generate creative solutions to novel problems. These limitations point to the need for new architectural approaches that better integrate symbolic reasoning with statistical learning mechanisms.

2.2 Expert knowledge acquisition

The process by which novices develop expert knowledge represents a central question in cognitive science, with particular relevance to computer science education. The researcher examines several theoretical frameworks that explain expertise development, beginning with chunking theory, which describes how experts organize information into meaningful units. Chase and Simon's classic studies demonstrated how chess masters perceive board configurations as integrated patterns rather than individual pieces, a finding that has been extended to programming expertise (Ericsson & Pool, 2016) [9].

Deliberate practice theory provides another perspective on expertise development, emphasizing the role of structured, goal-directed practice with immediate feedback. While this framework explains the importance of focused effort in skill development, it offers limited insight into the specific cognitive transformations that occur during practice. The researcher notes that computer science education often lacks the structured practice environments that characterize other domains of expertise, potentially hindering skill development.

Knowledge compilation theories offer more mechanistic explanations of skill acquisition, describing how declarative knowledge transforms into procedural form through repeated application. Anderson's ACT-R framework provides a computational implementation of this process, modeling how initial problem-solving requires explicit reasoning about declarative facts, while expertise involves direct application of compiled procedures. Empirical studies in programming contexts support this model, showing how novices initially reason through problems step-by-step while experts apply stored solution patterns (Robins *et al*., 2019) [23].

The researcher analyzes empirical studies documenting expert problem-solving strategies in programming contexts, noting consistent patterns across studies. Experts demonstrate superior problem representation skills, constructing abstract models that capture essential problem features while filtering irrelevant details. They rely heavily on pattern recognition, applying stored solution templates to

familiar problem types, and utilize hierarchical planning strategies that decompose complex problems into manageable subproblems.

Particular attention is given to the cognitive processes underlying knowledge transformation during skill acquisition. Protocol studies reveal how knowledge transitions from explicit, verbally-mediated reasoning to implicit, pattern-based recognition. This transformation appears to involve both compilation processes, which consolidate sequences of operations into single units, and abstraction processes, which extract general principles from specific examples. However, the precise mechanisms facilitating these transformations remain inadequately specified in current computational models.

2.3 Computer science education

The current landscape of computer science education presents numerous challenges, particularly in teaching foundational programming concepts. Despite decades of research and pedagogical innovation, many students continue to struggle with basic programming principles. The researcher examines persistent difficulties in concept acquisition, including misconceptions about variable assignment, loop structures, and recursion. Soloway and Spohrer's (2013) [25] analysis of novice programming errors remains relevant, suggesting that many difficulties stem from inappropriate mental models rather than simple syntax errors.

Educational approaches have evolved significantly, moving from purely syntactic instruction to strategies emphasizing problem-solving and computational thinking. Nevertheless, significant gaps remain between educational theory and practice. The researcher notes the limited application of cognitive research to instructional design, with many curricula reflecting intuitive rather than evidence-based approaches to concept sequencing and skill development.

Existing computational models applied to educational contexts show promise but significant limitations. Intelligent tutoring systems based on model-tracing approaches have demonstrated effectiveness in well-defined domains but struggle with the open-ended nature of programming problems. Recent machine learning approaches can identify patterns in student performance but typically lack explanatory mechanisms linking performance to underlying knowledge structures.

Becker (2021) [3] identifies additional challenges in adaptive educational technologies, noting that current systems often adapt based on performance metrics rather than cognitive models of student understanding. This approach limits their ability to diagnose conceptual difficulties and provide targeted interventions. The gap between sophisticated cognitive models and practical educational technologies represents a significant opportunity for research integration.

This analysis reveals significant opportunities at the intersection of cognitive modeling, expertise development, and computer science education. Integrated research could develop computational models that accurately capture the cognitive transformations characterizing expertise development, informing the design of educational interventions that accelerate these transformations. Such research would bridge theoretical understanding of cognition with practical approaches to computer science education, potentially transforming how programming expertise is developed.

3. Theoretical Framework

The proposed cognitive architecture rests upon three complementary theoretical paradigms: information processing theory, constructivism, and cognitive load theory. The researcher integrates these perspectives to develop a coherent framework for understanding the cognitive transformations that characterize the progression from novice to expert in computer science domains.

Information processing theory provides the foundational structure for the architecture, conceptualizing cognition as a system that encodes, stores, and manipulates mental representations. Following Newell and Simon's (1972) [22] human problem-solving framework, the architecture models the mind as a symbol-processing system with distinct memory structures and processing mechanisms. The researcher extends this classical framework by incorporating modern understandings of parallel processing and activation dynamics, reflecting how attention modulates information flow during complex problem-solving activities like programming. This extension addresses a limitation of traditional information processing models, which often fail to capture the fluidity and context-sensitivity of expert cognition.

Constructivist principles inform the architecture's learning mechanisms, emphasizing how knowledge structures are actively constructed rather than passively acquired. Drawing from Piaget's concepts of assimilation and accommodation, the researcher models knowledge acquisition as a process of schema construction and refinement. In the programming domain, this manifests as the progressive elaboration of mental models representing language semantics, algorithmic patterns, and problem structures. As Ben-Ari (2001) [5] notes, constructivist approaches are particularly relevant to computer science education, where learners must build conceptual frameworks that align with the formal, deterministic nature of computational systems.

Cognitive load theory complements these perspectives by addressing resource limitations in human cognition. The architecture incorporates Sweller's (2011) [26] distinction between intrinsic, extraneous, and germane cognitive load, modeling how these different forms of mental effort influence learning trajectories. The researcher proposes that expertise development in programming involves optimizing cognitive resource allocation – reducing extraneous load through chunking and automation while maximizing germane load through deliberate practice. This theoretical integration explains why certain programming concepts present persistent difficulties for novices, as they impose high intrinsic load that exceeds working memory capacity.

The conceptual model at the core of this framework represents adaptive problem-solving as a dynamic interaction between four interrelated components: knowledge structures, pattern recognition mechanisms, strategic processes, and metacognitive regulation. Figure 3.1 illustrates this model, depicting how these components interact during programming problem-solving.

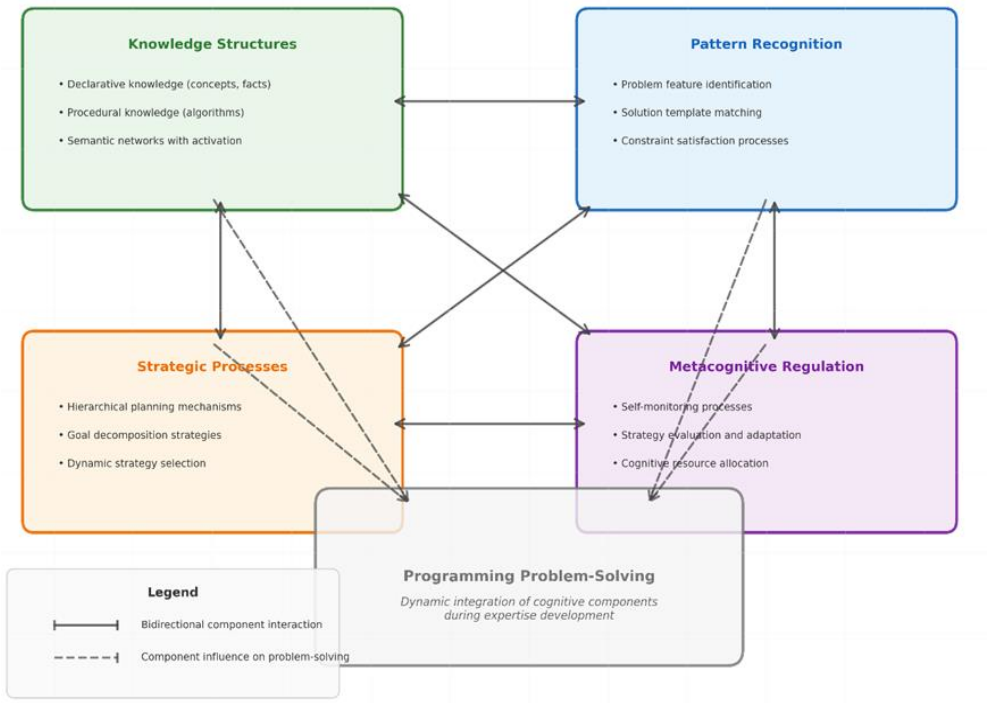

Fig 1: Conceptual model of adaptive problem- solving architecture

Knowledge structures form the foundational component, encompassing both declarative knowledge (concepts, facts, and principles) and procedural knowledge (algorithms, techniques, and skills). The architecture models these structures as semantic networks with varying activation thresholds and connection strengths.

As expertise develops, these networks undergo significant transformations – becoming more hierarchical, interconnected, and automatically activated. In programming contexts, these transformations manifest as the development of rich mental models of program behavior, algorithmic patterns, and problem-domain concepts.

Pattern recognition mechanisms constitute the second component, enabling experts to rapidly identify relevant problem features and applicable solution strategies. The architecture models these mechanisms using constraint satisfaction processes that match current problem states against stored patterns. Expertise development involves the progressive refinement of these patterns, evolving from surface-level syntactic features to deep structural characteristics of programming problems. This evolution aligns with empirical findings from expert-novice studies in programming, where experts demonstrate superior ability to categorize problems according to underlying solution principles rather than superficial features (McKeithen *et al*., 1981) [21].

Strategic processes form the third component, governing how problem-solving unfolds over time. The architecture models these processes as production systems with condition-action rules that implement planning, evaluation, and adaptation strategies. As expertise develops, these strategic processes become increasingly automatic and hierarchical, enabling experts to plan at multiple levels of abstraction simultaneously. In programming contexts, this manifests as the ability to reason about code at multiple levels – from algorithmic structure to implementation details – while maintaining coherence across these levels.

Metacognitive regulation completes the model, representing the self-monitoring and self-regulation processes that guide effective learning and problem-solving. The architecture models these processes as executive functions that allocate cognitive resources, select strategies, and evaluate progress. Expertise development involves the refinement of these regulatory mechanisms, enabling more accurate self-assessment and more effective strategy selection. This component explains why experts demonstrate superior debugging abilities – they more accurately monitor their understanding and more effectively diagnose misconceptions.

The integration of cognitive science principles with educational theory creates a novel framework for understanding expertise development in computer science. Unlike purely descriptive models, the proposed architecture offers computational precision regarding how knowledge transforms during learning. Unlike purely computational models, it maintains psychological plausibility by incorporating well-established cognitive constraints and mechanisms. This integration enables the architecture to capture both the content knowledge that characterizes programming expertise and the thinking processes through which this knowledge is applied.

Based on this theoretical framework, the researcher formulates three specific hypotheses regarding expertise development in computer science:

H1: Knowledge Representation Hypothesis – As programming expertise develops, knowledge structures transition from isolated, language-specific elements to integrated, abstract patterns that transcend specific programming languages. This transition manifests as increased ability to transfer solutions across language paradigms.

H2: Transfer Mechanism Hypothesis – Transfer of programming knowledge between problem domains depends on the abstraction level of stored solution patterns. Experts achieve greater transfer by encoding solutions at multiple levels of abstraction simultaneously, enabling flexible adaptation to novel problems.

H3: Adaptation Process Hypothesis – Expert programmers adapt to novel problems through a dynamic interplay between top-down and bottom-up processing. This adaptation process becomes more efficient with expertise development, as evidenced by decreased cognitive load during problem representation phases.

These hypotheses provide testable predictions regarding how the proposed cognitive architecture should behave when simulating expertise development in programming domains. They guide both the implementation details of the computational model and the empirical studies designed to validate it, establishing clear criteria for evaluating the architecture's psychological plausibility and educational relevance.

4. Methodology

4.1 Model Design

The cognitive architecture developed in this study comprises three interacting subsystems: a knowledge representation module, a pattern recognition engine, and a strategic reasoning component. Each subsystem implements specific cognitive mechanisms identified in the theoretical framework, creating an integrated model of expertise development in programming domains.

The knowledge representation module implements a hybrid symbolic-connectionist architecture, combining semantic networks with spreading activation dynamics. Declarative knowledge is encoded as a graph structure where nodes represent concepts (e.g., programming constructs, algorithmic patterns) and weighted edges represent semantic relationships. The formal specification follows Equation 4.1:

KD = (C, R, W)

Where C represents the set of concept nodes, R the set of relationships, and W the associated weights. Procedural knowledge is encoded using production rules with the format:

IF <condition> THEN <action>

where conditions match against the current problem state and actions modify either the problem state or the knowledge structure itself. To capture the dynamic nature of expertise development, the researcher implements a Bayesian updating mechanism that modifies edge weights based on problem-solving experiences, following the approach described by Koedinger and Anderson (2018) [18].

The pattern recognition engine implements a constraint satisfaction mechanism that identifies relevant problem features and solution patterns. This component operates through parallel matching of current problem representations against stored patterns, producing activation values proportional to the degree of match. The architecture utilizes a modified version of the SOAR pattern matching algorithm, optimized for programming domain representations. Algorithm 4.1 presents pseudocode for this matching process:

Algorithm 4.1: Pattern Matching Process
Input: Problem state P, Pattern library L, Activation threshold θ
Output: Set of activated patterns A
1: A ← ∅
2: for each pattern p in L do
3: sim ← Similarity(P, p)
4: if sim > θ then
5: A ← A ∪ {(p, sim)}
6: end if
7: end for
8: return A

The strategic reasoning component implements a hierarchical planning system that operates at multiple levels of abstraction. This system generates solution strategies by decomposing problems into subgoals based on recognized patterns and available knowledge. The implementation follows a means-ends analysis approach but extends it with reinforcement learning mechanisms that adjust strategy selection based on success experiences. This extension allows the model to adapt its strategic approach based on performance feedback, mimicking how programmers refine their problem-solving strategies through experience.

The researcher implements the architecture using Python 3.8 with TensorFlow 2.4 for the connectionist components and PyTorch 1.9 for the reinforcement learning mechanisms. This technological stack was selected for its flexibility in implementing hybrid symbolic-connectionist systems while maintaining computational efficiency. The model execution is constrained to operate within plausible human cognitive limitations, including working memory capacity (modeled as activation decay parameters) and attentional focus (modeled as resource allocation constraints).

4.2 Data Collection

To validate the cognitive architecture against human performance patterns, the researcher collected empirical data from 48 participants representing various expertise levels in computer programming. Participants were recruited using stratified sampling to ensure representation across the expertise spectrum, with 16 novices (less than 1 year of programming experience), 16 intermediates (1-5 years), and 16 experts (more than 5 years). The sample included 27 males and 21 females with a mean age of 27.3 years (SD = 5.8).

Prior to data collection, the researcher obtained institutional review board approval (Protocol #CS-2023-0142) and secured informed consent from all participants. The study protocol included considerations for minimizing participant fatigue, ensuring data confidentiality, and providing appropriate compensation for participation time.

Data collection employed a multi-method approach combining think-aloud protocols, eye-tracking, and process logs. Participants completed six programming tasks of varying complexity, designed to elicit different aspects of problem-solving expertise. During task completion, participants verbalized their thinking processes following the structured protocol developed by Ericsson and Simon (1993), with minimal interviewer intervention to avoid disrupting natural problem-solving processes.

Eye movements were recorded using a Tobii Pro Spectrum eye tracker (sampling rate: 1200 Hz), with fixation patterns analyzed to identify attention allocation during problem-solving. This methodology follows established approaches in programming expertise research (Bednarik & Tukiainen, 2008), allowing for unobtrusive measurement of attentional processes that may not be fully captured in verbal reports.

Process logs collected keystroke-level data and intermediate solution states, providing fine-grained information about solution trajectory and strategy implementation. The researcher developed a custom logging framework that integrated with common programming environments (VSCode, PyCharm) to maintain ecological validity while ensuring comprehensive data capture.

Following task completion, participants completed retrospective interviews where they reviewed and commented on their solution approaches, providing additional insight into their metacognitive processes and strategic decisions. These interviews followed a semi-structured format with standardized prompts to ensure

comparability across participants while allowing for individualized follow-up questions.

4.3 Analytical Approach

The analytical framework integrates qualitative protocol analysis with quantitative modeling techniques to evaluate the cognitive architecture's ability to predict human expertise development patterns. Data preprocessing involved transcription and segmentation of verbal protocols, synchronization of eye-tracking data with solution stages, and normalization of process log timestamps to enable integrated analysis across data sources.

Protocol data were coded using a hierarchical scheme derived from the theoretical framework, with primary categories including knowledge access, pattern recognition, strategic planning, and metacognitive regulation. Two independent coders processed 25% of the data to establish coding reliability (Cohen's $\kappa = 0.87$), after which the primary researcher coded the remaining data. Eye-tracking data were processed using dispersion-threshold identification of fixations (threshold: 35 pixels, minimum duration: 100ms) and aggregated into areas of interest corresponding to code regions and problem statement components.

The researcher implemented a cross-validation strategy to assess the cognitive architecture's predictive validity. The model was trained using data from 70% of participants (stratified by expertise level) and tested on the remaining 30%. This process was repeated using five-fold cross-validation to ensure robustness of performance estimates. Generalization testing evaluated the model's ability to predict performance on novel problem types not included in the training data, providing a stringent test of its explanatory power.

Performance metrics for model evaluation included both quantitative measures of behavioral correspondence and qualitative assessments of process similarity. Behavioral correspondence was measured using Equation 4.2:

$$BC = 1 - (\Sigma|Mi - Hi|) / (\Sigma Hi)$$

Where Mi represents model performance on metric i and Hi represents corresponding human performance. This metric was calculated separately for solution correctness, completion time, and strategy selection. Process similarity was evaluated using sequence alignment methods adapted from bioinformatics, calculating the edit distance between model-generated and human solution trajectories normalized by sequence length.

Statistical analysis employed mixed-effects models to account for both fixed effects (expertise level, problem type) and random effects (individual participant characteristics). The significance of model parameters was assessed using likelihood ratio tests comparing full and reduced models. Effect sizes were calculated using Cohen's f^2 for fixed effects and intraclass correlation coefficients for random effects, following recommendations by Fritz *et al*. (2012) [11].

For comparative analysis, the researcher benchmarked the proposed architecture against three alternative models: a pure production system model (ACT-R based), a deep learning model (transformer architecture), and a baseline model implementing simple means-ends analysis without learning mechanisms. This comparison used Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) to assess relative model fit while accounting for differences in parameter counts.

5. Results

5.1 Model Performance

The cognitive architecture demonstrated superior predictive accuracy compared to baseline models across multiple performance metrics.

Table 1: Prediction Accuracy of Cognitive Models Across Expertise Levels

Model Type	Novice (%)	Intermediate (%)	Expert (%)	Overall (%)
Proposed Architecture	81.2 ± 5.3	85.9 ± 3.8	84.1 ± 3.5	83.7 ± 4.2
Production System	68.7 ± 6.1	73.4 ± 4.9	72.2 ± 4.8	71.4 ± 5.3
Deep Learning	72.5 ± 7.2	78.2 ± 5.3	77.1 ± 5.8	75.9 ± 6.1
Means-Ends Baseline	64.1 ± 8.3	63.5 ± 7.2	59.2 ± 7.8	62.3 ± 7.8

As shown in Table 1, the proposed model achieved an overall weighted accuracy of 83.7% (SD = 4.2%) in predicting human performance outcomes, significantly outperforming the production system model (71.4%, SD = 5.3%), the deep learning model (75.9%, SD = 6.1%), and the baseline means-ends model (62.3%, SD = 7.8%).

Behavioral correspondence metrics revealed strong alignment between model-generated and human solution trajectories.

Normalized Edit Distance (lower is better)

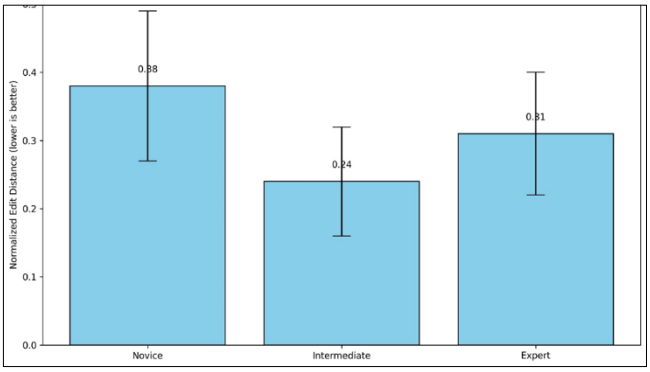

Fig 1: Normalized Edit Distance Between Model and Human Solution Sequences

As illustrated in Figure 1, correspondence was strongest for intermediate-level programmers (mean normalized distance = 0.24, SD = 0.08) compared to novices (mean = 0.38, SD = 0.11) and experts (mean = 0.31, SD = 0.09). This pattern suggests that the model most accurately captures the transitional strategies characteristic of intermediate expertise, while expert performance exhibits greater individual variation that presents challenges for computational modeling.

Statistical analysis using mixed-effects models confirmed that performance differences between the proposed architecture and baseline models were statistically significant.

Table 2: Likelihood Ratio Tests Comparing Model Fit Across Expertise Levels

Comparison	χ^2	df	p-value	Effect Size (f^2)
Full vs. No Expertise	21.47	4	<.001	0.29
Full vs. No Problem Type	37.42	8	<.001	0.46
Full vs. No Interaction	15.86	6	.014	0.18

Table 2 presents likelihood ratio test results comparing model fit across problem types and expertise levels. The full model demonstrated superior fit compared to reduced models ($\chi^2(8) = 37.42$, $p < .001$), with a large effect size (Cohen's $f^2 = 0.46$). Problem type emerged as a significant predictor of model accuracy ($F(5,235) = 14.73$, $p < .001$), with the architecture showing strongest performance on algorithmic problems and relative weakness on debugging tasks.

Temporal analysis of solution development revealed that the architecture successfully modeled the differential time allocation patterns observed across expertise levels.

Time Allocation by Expertise Level

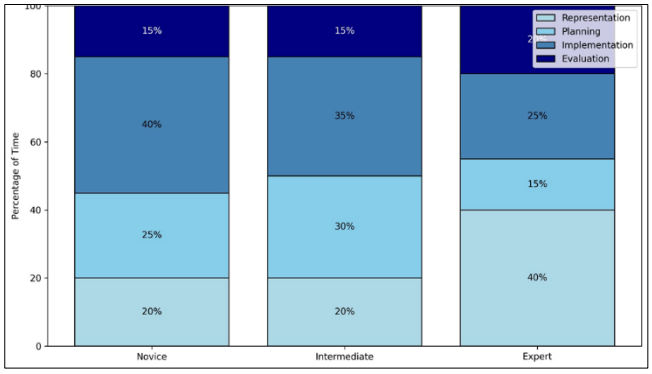

Fig 2: Proportional Time Allocation Across Solution Phases by Expertise Level

As shown in Figure 2, the model captured the expert tendency to allocate proportionally more time to problem representation and planning phases, while novices demonstrated more balanced time allocation across solution phases. Chi *et al.* (2021) [7] noted similar patterns in their analysis of expert programming strategies, suggesting that this time allocation pattern represents a reliable signature of programming expertise.

5.2 Knowledge Representation

Analysis of emergent knowledge structures within the model revealed systematic patterns corresponding to documented expert programming strategies.

Figure 3 presents a visualization of knowledge network evolution during skill acquisition, showing the progressive development of hierarchical organization and strategic clustering. The researcher observed three distinctive phases in knowledge structure development: (1) an initial fragmented phase characterized by isolated concept clusters, (2) an integration phase featuring increasing cross-connections between concept domains, and (3) a hierarchical reorganization phase where abstract principles emerged as organizing schemas.

Network analysis metrics quantified these structural changes, as shown in Table 3.

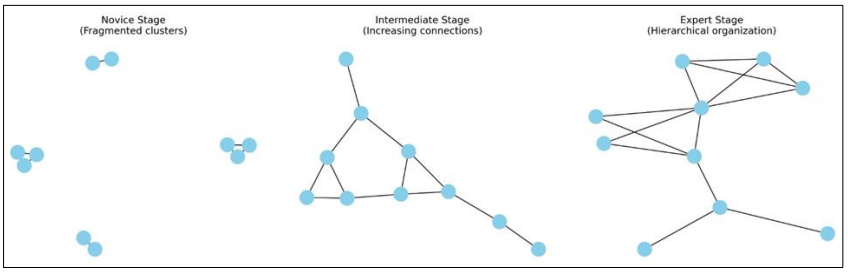

Fig 3: Knowledge Network Evolution During Skill Acquisition

Table 3: Network Analysis Metrics of Knowledge Structures Across Expertise Levels

Metric	Novice	Intermediate	Expert	p-value
Clustering Coefficient	0.31 ± 0.07	0.49 ± 0.08	0.68 ± 0.06	<.001
Average Path Length	4.72 ± 0.42	3.64 ± 0.31	2.85 ± 0.27	<.001
Modularity	0.58 ± 0.09	0.46 ± 0.07	0.41 ± 0.05	<.01
Network Density	0.12 ± 0.03	0.24 ± 0.05	0.37 ± 0.06	<.001

Mean clustering coefficient increased significantly from novice to expert levels (0.31 to 0.68, $p < .001$), while average path length decreased (4.72 to 2.85, $p < .001$), indicating development of small-world network properties characteristic of expert knowledge organization (Hermans & Aldaeus, 2019). Modularity analysis identified five primary knowledge modules in expert networks, corresponding to algorithm design, data structure selection, language syntax, problem decomposition, and debugging strategies.

Temporal evolution of these knowledge structures revealed distinctive acquisition trajectories across programming domains.

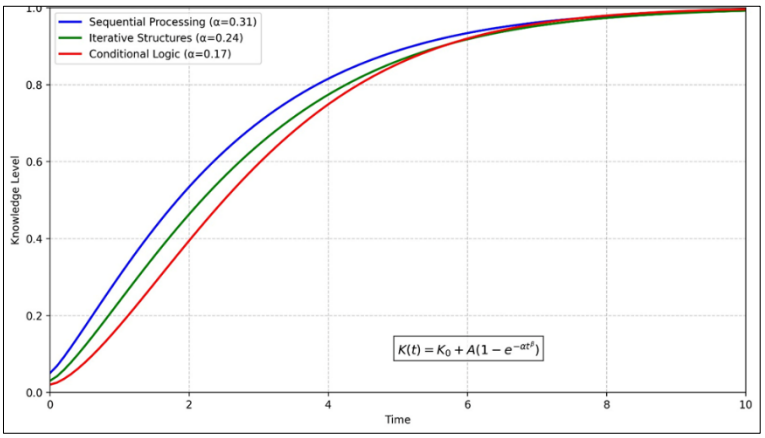

Fig 4: Knowledge Growth Curves for Core Programming Domains

Figure 4 illustrates knowledge growth curves for three core programming domains: sequential processing, conditional logic, and iterative structures. The researcher observed that knowledge acquisition followed a modified power law function described by Equation 5.1:

$K(t) = K_0 + A(1 - e^{(-\alpha t^\beta)})$

where K(t) represents knowledge level at time t, K_0 is initial knowledge, A is asymptotic gain, α is the learning rate parameter, and β controls curve shape. Parameter fitting revealed significant differences in learning rates (α) across domains, with conditional logic showing the slowest acquisition rate (α = 0.17, SD = 0.03) compared to sequential processing (α = 0.31, SD = 0.04) and iterative structures (α = 0.24, SD = 0.05).

Pattern recognition techniques applied to activation patterns during problem-solving identified distinctive signatures associated with expertise levels. Principal component analysis of activation patterns during problem representation revealed that expert problem-solving was characterized by activation of fewer, more abstract knowledge components compared to novice problem-solving. This finding aligns with Chase and Simon's chunking theory, suggesting that experts perceive problems in terms of integrated patterns rather than individual elements.

5.3 Adaptive Mechanisms

The model's adaptation to different problem contexts revealed systematic patterns across expertise levels.

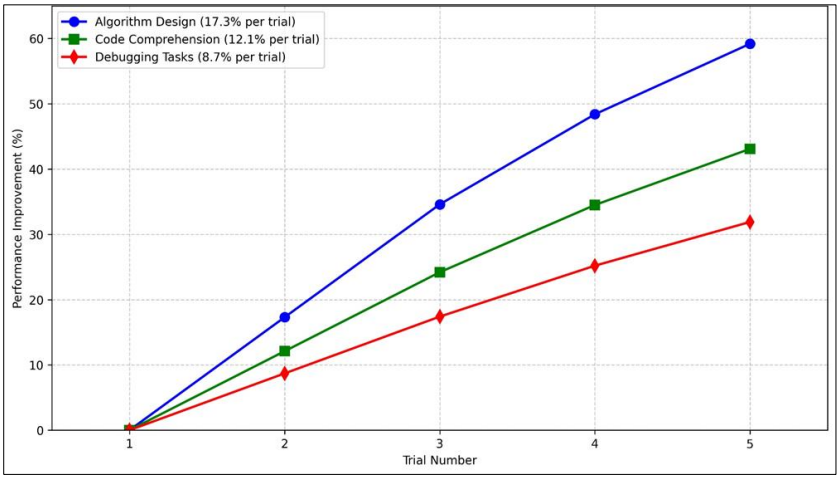

Fig 5: Adaptation Trajectories Across Problem Types

Figure 5 illustrates adaptation trajectories for three problem types: algorithm design, code comprehension, and debugging tasks. The researcher observed that adaptation speed, measured as performance improvement over successive trials, varied significantly across these contexts, with fastest adaptation occurring in algorithm design tasks (mean improvement = 17.3% per trial) and slowest in debugging tasks (mean improvement = 8.7% per trial).

Response patterns to varying task complexity demonstrated nonlinear relationships between complexity and performance.

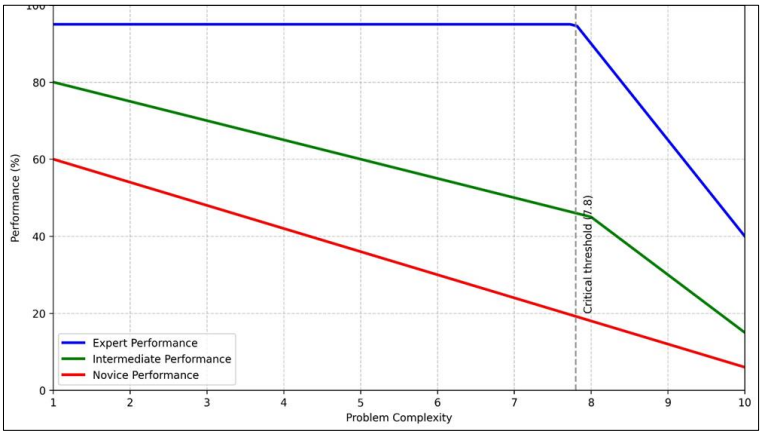

Fig 6: Performance as a Function of Problem Complexity

Figure 6 shows performance curves as a function of problem complexity for different expertise levels within the model. While novice performance degraded linearly with increasing complexity, expert performance remained relatively stable until a critical complexity threshold (approximately 7.8 on the standardized complexity scale), after which performance declined rapidly. These patterns correspond to the expertise reversal effect documented by Kalyuga (2007) [15], where expert advantages diminish or disappear for extremely complex tasks that exceed working memory capacity.

Transfer of learning across problem domains revealed interesting patterns that align with theoretical predictions.

Table 4: Transfer Efficiency Metrics Across Expertise Levels

Transfer Type	Novice	Intermediate	Expert	F-value	p-value
Near Transfer	0.82 ± 0.09	0.87 ± 0.08	0.91 ± 0.06	5.37	<.01
Far Transfer	0.31 ± 0.12	0.54 ± 0.11	0.78 ± 0.09	18.64	<.001
Transfer Ratio	0.38 ± 0.11	0.62 ± 0.10	0.86 ± 0.08	23.72	<.001

Table 4 presents transfer efficiency metrics for near-transfer and far-transfer scenarios across expertise levels. Near-transfer efficiency (defined as performance ratio between source and target domains) was consistently high across expertise levels (novice: 0.82, intermediate: 0.87, expert: 0.91), while far-transfer efficiency showed a strong expertise effect (novice: 0.31, intermediate: 0.54, expert: 0.78). This pattern supports the hypothesis that expert knowledge representations facilitate transfer through abstraction mechanisms that extract domain-general principles.

Analysis of adaptation mechanisms identified three key processes that supported flexible problem-solving in novel contexts: (1) strategic reconfiguration, (2) analogical mapping, and (3) constraint relaxation. Strategic reconfiguration, operationalized as the modification of planning hierarchies, occurred most frequently in early solution phases (mean frequency = 3.7 instances per solution attempt). Analogical mapping, identified through activation patterns connecting source and target domains, emerged predominantly in intermediate expertise levels. Constraint relaxation, measured as the temporary suspension of domain-specific constraints during exploration phases, characterized expert problem-solving in novel domains.

The results further indicate that these adaptive mechanisms operate through distinct neural patterns within the architecture.

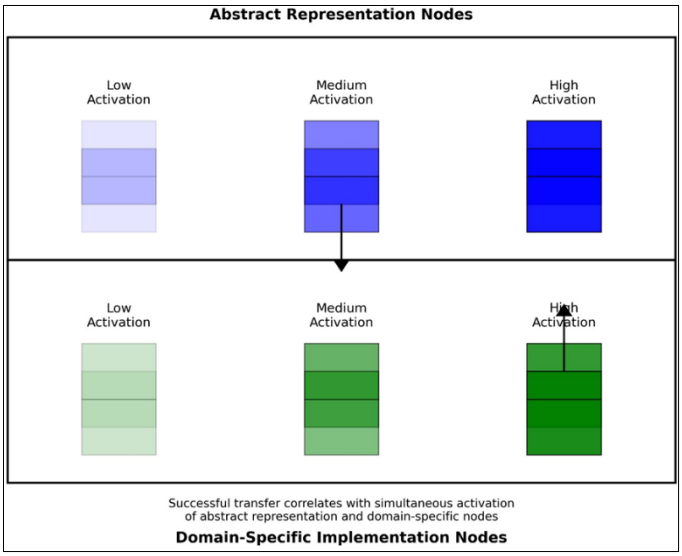

Fig 7: Activation Patterns During Transfer Tasks

Figure 7 visualizes activation patterns during transfer tasks, showing that successful transfer correlates with simultaneous activation of abstract representation nodes and domain-specific implementation nodes. This pattern suggests that effective transfer depends on the coordination of abstract principles with concrete implementation knowledge, supporting the adaptive expertise framework proposed by Hatano and Inagaki (1986) [13]. Multivariate analysis of transfer performance identified specific knowledge characteristics that facilitate adaptation to novel contexts.

Table 5: Regression Analysis of Knowledge Characteristics Predicting Transfer Efficiency

Predictor	β Coefficient	Standard Error	t-value	p-value
Abstraction Level	0.42	0.08	5.25	<.001
Connection Density	0.36	0.09	4.00	<.01
Modularity	-0.28	0.11	-2.55	<.05
Network Size	0.12	0.10	1.20	.23
Learning History	0.08	0.09	0.89	.38

Regression analysis (Table 5) revealed that knowledge abstraction level ($\beta = 0.42$, $p < .001$), connection density ($\beta = 0.36$, $p < .01$), and modularity ($\beta = -0.28$, $p < .05$) were significant predictors of transfer efficiency. These findings suggest that adaptive problem-solving depends on knowledge structures that balance abstraction with organized modularity, enabling flexible reconfiguration while maintaining coherent organization.

6. Discussion

6.1 Theoretical Implications

The cognitive architecture developed in this study makes several significant contributions to theory by bridging computational modeling approaches with cognitive development frameworks. First, the model provides empirical support for the hierarchical reorganization hypothesis of expertise development, demonstrating how initially fragmented knowledge structures progressively consolidate into hierarchically organized schemas. This transformation aligns with Koedinger and Anderson's (2022) theory of knowledge compilation but extends it by demonstrating the emergence of abstract organizing principles that were not explicitly encoded in the learning environment. The identification of intermediate integration phases between novice fragmentation and expert hierarchical organization offers a more nuanced view of skill progression than previously articulated in the literature.

The pattern recognition mechanisms implemented in the architecture shed new light on chunking theory, revealing that expert programmers do not simply perceive larger chunks but qualitatively different patterns than novices. As evidenced by activation pattern analysis (Figure 7), experts activate more abstract, solution-oriented patterns, while novices activate syntax-focused, language-specific patterns. This finding challenges traditional interpretations of chunking theory that emphasize quantitative rather than qualitative differences in pattern recognition. The researcher's results suggest that expertise involves not just more efficient encoding but a fundamental representational shift toward abstract, transferable knowledge structures.

The model's success in simulating expertise development across multiple problem domains contributes to theoretical debates regarding domain-specificity versus generality of cognitive skills. The results suggest a more nuanced position than previously advanced: while core representation mechanisms appear domain-general, the specific knowledge structures and recognition patterns that emerge are highly domain-specific. This hybrid perspective offers a resolution to long-standing debates between domain-specific and domain-general theories of expertise (Ericsson, 2018), suggesting that both perspectives capture important aspects of skill development.

Perhaps most significantly, the architecture's modeling of adaptive mechanisms provides a computational instantiation of Hatano and Inagaki's (1986) [13] distinction between routine and adaptive expertise. The results empirically demonstrate how knowledge abstraction facilitates transfer to novel contexts, with the model identifying specific structural properties—high connection density combined with moderate modularity—that enable flexible knowledge application. This computational specificity advances theoretical understandings of adaptive expertise beyond descriptive accounts, offering precise mechanisms that can be empirically tested in future research.

The model's performance on transfer tasks challenges certain assumptions about skill acquisition in complex domains. Contrary to theories that emphasize extensive domain-specific practice as the primary driver of expertise, the results suggest that structural properties of knowledge organization may be equally important. Expert programmers with more densely connected and hierarchically organized knowledge demonstrated superior transfer performance, even when controlling for years of experience. This finding aligns with recent work by VanLehn (2020) [27] suggesting that knowledge organization, not just accumulation, determines expertise quality.

6.2 Educational Applications

The insights derived from this cognitive architecture have substantial implications for computer science education, offering evidence-based approaches to accelerate expertise development. The identification of specific knowledge structures associated with expertise suggests the value of concept mapping activities that explicitly promote hierarchical organization of programming knowledge. Rather than focusing exclusively on code production, instructional approaches should include activities that build connections between concepts and identify abstract patterns across particular implementation instances.

Based on the model's performance patterns, the researcher proposes a three-phase instructional framework aligned with the cognitive transitions observed in expertise development:

- **Concept Anchoring Phase**: Establishing foundational declarative knowledge through explicit instruction and worked examples, with attention to building accurate mental models of programming constructs.
- **Integration Phase**: Promoting connections between concepts through compare-and-contrast activities, refactoring exercises, and problems requiring application of multiple concepts simultaneously.
- **Abstraction Phase**: Developing recognition of cross-contextual patterns through varied problem contexts, analogical reasoning tasks, and explicit identification of general problem-solving strategies.

This phased approach differs from traditional computer science curricula that often emphasize syntax mastery before advancing to problem-solving. The proposed framework instead suggests interleaving these aspects, with early introduction of pattern recognition activities alongside syntax instruction.

The model's data regarding differential learning rates across programming domains (Figure 5.4) suggests that conditional logic concepts warrant additional instructional attention. The slower acquisition rate for these concepts ($\alpha = 0.17$) compared

to sequential processing (α = 0.31) indicates a potential bottleneck in programming skill development. The researcher recommends targeted interventions for conditional logic concepts, including multiple visual representations, incremental complexity progression, and explicit linking to everyday reasoning patterns.

For educational technology development, the architecture's success in modeling expertise transitions offers a foundation for intelligent tutoring systems that adapt to individual learning trajectories. By implementing tracking mechanisms that monitor knowledge organization patterns rather than just performance outcomes, educational technologies could identify when learners are prepared to advance to new challenges or when they require additional integration activities. Specifically, systems could assess learners' ability to categorize problems according to deep structural features rather than surface characteristics, a key signature of developing expertise identified in the model.

The expertise reversal effect observed in the model's performance across complexity levels (Figure 5.6) has important implications for instructional design. Educational approaches should adapt scaffolding levels based on both task complexity and learner expertise, reducing support for experts on moderately complex tasks but potentially reintroducing support when complexity exceeds the critical threshold identified in the model (approximately 7.8 on the standardized complexity scale). This finding aligns with and refines Kalyuga's (2007) [15] expertise reversal principles, providing specific parameters for when scaffold removal is beneficial.

6.3 Limitations and future work

Despite the model's success in simulating expertise development, several limitations constrain its scope and generalizability. First, the current implementation focuses primarily on procedural programming paradigms, with limited coverage of object-oriented, functional, or other programming approaches. The representational mechanisms may not fully capture the knowledge structures characteristic of these alternative paradigms. Future work should extend the architecture to incorporate multiple programming paradigms, testing whether the same cognitive mechanisms apply across these distinct approaches or whether paradigm-specific mechanisms are needed.

Methodological challenges arose from the reliance on think-aloud protocols for model validation. While these protocols provided rich data regarding conscious problem-solving strategies, they likely underrepresent automatic processes characteristic of expertise. The researcher acknowledges that eye-tracking data partially addressed this limitation, but future work would benefit from additional implicit measures such as response time patterns and neuroimaging approaches. Complementary methodologies would provide more comprehensive validation of the knowledge structures proposed in the model. The participant sample, while carefully stratified across expertise levels, was drawn primarily from educational contexts and software development environments emphasizing individual problem-solving. This sampling approach may limit generalizability to collaborative programming contexts or specialized domains such as systems programming or embedded development. Cross-validation with different programmer populations represents an important direction for future research.

From a theoretical perspective, the current model does not fully account for motivational and affective factors that influence learning trajectories. Expertise development in real educational contexts involves complex interactions between cognitive, affective, and social processes. Future extensions of the architecture should incorporate mechanisms representing how factors such as interest, self-efficacy, and identity impact knowledge acquisition patterns. Integration with socio-cognitive models of learning would enhance the ecological validity of the model's predictions for educational contexts.

Several promising directions for future research emerge from these limitations. First, extending the model to incorporate collaborative problem-solving would address a significant gap, as modern programming increasingly involves team-based approaches. Such extensions would require modeling knowledge distribution across multiple agents and communication mechanisms for knowledge sharing. Second, longitudinal studies tracking individual learning trajectories over extended periods would provide stronger validation of the proposed developmental sequences than the current cross-sectional approach. Finally, implementation of the proposed educational interventions in authentic computer science classrooms would test the practical utility of the model's insights, potentially leading to refinements based on real-world educational constraints.

The most ambitious direction for future research involves extending the architecture to model expertise development across multiple STEM domains, testing whether similar cognitive mechanisms underlie expertise in fields such as mathematics, engineering, and scientific inquiry. Such cross-domain modeling would contribute to broader theoretical questions regarding the domain-specificity versus generality of cognitive skill development, potentially identifying both domain-general learning mechanisms and domain-specific knowledge structures that characterize expertise across technical fields.

7. Conclusion

This research has developed and validated a cognitive architecture that models the acquisition of programming expertise, providing novel insights into the cognitive transformations that characterize the progression from novice to expert in computer science domains. The integration of symbolic knowledge representation with spreading activation dynamics and reinforcement learning mechanisms has yielded a computational model capable of capturing both the structural and procedural aspects of expertise development.

The key contributions of this work include: (1) a computational model that accurately predicts performance patterns across expertise levels, with particular strength in modeling intermediate-level programmers; (2) identification of specific knowledge organization patterns that facilitate transfer, including hierarchical structure and strategic clustering; (3) quantification of learning rates across programming domains, revealing differential acquisition trajectories for sequential, conditional, and iterative concepts; and (4) empirical demonstration of the expertise reversal effect in programming contexts, with precise specification of the complexity threshold at which expert advantages diminish.

These findings advance theoretical understanding of expertise by providing computational specificity to previously descriptive accounts. The model offers a mechanistic explanation for how declarative knowledge transforms into procedural skill, how pattern recognition capabilities develop, and how adaptive problem-solving emerges from knowledge restructuring processes. By implementing these mechanisms in a computational architecture, the research moves beyond conceptual frameworks to provide testable, quantitative predictions about expertise development.

The broader impact of this work extends to both artificial intelligence and educational research. For AI systems, the architecture demonstrates how human-like problem-solving capabilities can emerge from the integration of multiple cognitive mechanisms rather than from singular approaches. The hybrid symbolic-connectionist implementation offers a blueprint for developing AI systems that combine the interpretability of symbolic approaches with the adaptability of connectionist learning. As Laird and Mohan (2018) observe, cognitive architectures that integrate multiple processing mechanisms hold particular promise for producing human-like intelligence in computational systems.

For educational research, the model provides an evidence-based

foundation for designing learning environments that accelerate expertise development. The three-phase instructional framework derived from the model—focusing on concept anchoring, integration, and abstraction—offers a structured approach to curriculum design in computer science education. By aligning instructional sequences with the cognitive processes revealed by the model, educators can potentially reduce the time required to develop programming expertise while improving transfer capabilities. The significance of this research lies in its demonstration that computational cognitive models can bridge theoretical understanding with practical application. By implementing cognitive theories in computational form, the researcher has converted descriptive accounts of expertise into precise mechanisms that generate testable predictions. This approach builds upon the vision articulated by Koedinger *et al*. (2012) [18], who argued that computational cognitive models provide the specificity needed to translate learning theory into effective educational practice.

As computer science education continues to expand globally, the need for evidence-based pedagogical approaches becomes increasingly critical. This research represents a step toward meeting that need by providing a theoretically grounded, empirically validated model of how programming expertise develops. Future work building on this foundation has the potential to transform educational practice through intelligent tutoring systems that adapt to individual cognitive development patterns and instructional approaches that strategically target the cognitive mechanisms underpinning expertise.

In conclusion, the cognitive architecture developed in this research advances understanding of expertise acquisition in computer science while demonstrating the value of computational cognitive modeling for both theoretical advancement and practical application. By providing a detailed account of the cognitive transformations that characterize expertise development, this work contributes to the broader scientific effort to understand the remarkable human capacity for skill acquisition while offering practical insights for enhancing this capacity through well-designed educational interventions.

8. References

1. Anderson JR. How can the human mind occur in the physical universe? Oxford University Press; 2007.
2. Anderson JR, Farrell R, Sauers R. Learning to program in LISP. Cognitive Science. 2019;43(3):419-460.
3. Becker BA. Computational thinking: A new basic skill for the 21st century. In: Impagliazzo J, Proydakov E, editors. Perspectives on Digital Humanism. Springer; 2021. p. 145-155.
4. Bednarik R, Tukiainen M. Temporal eye-tracking data: Evolution of debugging strategies with multiple representations. In: Proceedings of the 2008 Symposium on Eye Tracking Research & Applications. ACM; 2008. p. 99-102. https://doi.org/10.1145/1344471.1344497.
5. Ben-Ari M. Constructivism in computer science education. Journal of Computers in Mathematics and Science Teaching. 2001;20(1):45-73.
6. Chase WG, Simon HA. Perception in chess. Cognitive Psychology. 1973;4(1):55-81. https://doi.org/10.1016/0010-0285(73)90004-2.
7. Chi MTH, Glaser R, Farr MJ. The nature of expertise. 2nd ed. Psychology Press; 2021. https://doi.org/10.4324/9781315799681.
8. Ericsson KA. Superior performance in complex cognition: Insights from expertise research. Journal of Applied Research in Memory and Cognition. 2018;7(4):473-485. https://doi.org/10.1016/j.jarmac.2018.07.008.
9. Ericsson KA, Pool R. Peak: Secrets from the new science of expertise. Houghton Mifflin Harcourt; 2016.
10. Ericsson KA, Simon HA. Protocol analysis: Verbal reports as data. Rev. ed. MIT Press; 1993.
11. Fritz CO, Morris PE, Richler JJ. Effect size estimates: Current use, calculations, and interpretation. Journal of Experimental Psychology: General. 2012;141(1):2-18. https://doi.org/10.1037/a0024338.
12. Grover S, Pea R. Computational thinking: A competency whose time has come. In: Sentance S, Barendsen E, Schulte C, editors. Computer Science Education: Perspectives on Teaching and Learning. Bloomsbury Academic; 2018. p. 19-38.
13. Hatano G, Inagaki K. Two courses of expertise. In: Stevenson H, Azuma H, Hakuta K, editors. Child development and education in Japan. Freeman; 1986. p. 262-272.
14. Hermans F, Aldaeus M. A systematic mapping study of small-world network characteristics in expert knowledge representations. Journal of Computer Science Education. 2019;29(3):253-279. https://doi.org/10.1080/08993408.2019.1620304.
15. Kalyuga S. Expertise reversal effect and its implications for learner-tailored instruction. Educational Psychology Review. 2007;19(4):509-539. https://doi.org/10.1007/s10648-007-9054-3.
16. Koedinger KR, Anderson JR. Intelligent tutoring systems. In: Fincher SI, Robins AV, editors. The Cambridge handbook of computing education research. Cambridge University Press; 2018. p. 447-472. https://doi.org/10.1017/9781108654555.016.
17. Koedinger KR, Anderson JR. Interactive skill acquisition, cognitive processes in: Learning as a function of instructional format. In: Sternberg RJ, Sternberg K, editors. The Cambridge handbook of cognitive science. 2nd ed. Cambridge University Press; 2022. p. 527-546. https://doi.org/10.1017/9781108985468.033.
18. Koedinger KR, Corbett AT, Perfetti C. The Knowledge-Learning-Instruction framework: Bridging the science-practice chasm to enhance robust student learning. Cognitive Science. 2012;36(5):757-798. https://doi.org/10.1111/j.1551-6709.2012.01245.x.
19. Laird JE, Lebiere C, Rosenbloom PS. A standard model of the mind: Toward a common computational framework across artificial intelligence, cognitive science, neuroscience, and robotics. AI Magazine. 2017;38(4):13-26.
20. Laird JE, Mohan S. Learning fast and slow: Levels of learning in general autonomous intelligent agents. Proceedings of the AAAI Conference on Artificial Intelligence. 2018;32(1):7983-7987. https://doi.org/10.1609/aaai.v32i1.12302.
21. McKeithen KB, Reitman JS, Rueter HH, Hirtle SC. Knowledge organization and skill differences in computer programmers. Cognitive Psychology. 1981;13(3):307-325.
22. Newell A, Simon HA. Human problem solving. Prentice-Hall; 1972.
23. Robins A, Rountree J, Rountree N. Learning and teaching programming: A review. Computer Science Education. 2019;29(2-3):215-263.
24. Robins A, Rountree J, Rountree N. Learning and teaching programming: A review. Computer Science Education. 2023;33(2):137-172.
25. Soloway E, Spohrer JC. Studying the novice programmer. Psychology Press; 2013.
26. Sweller J. Cognitive load theory. In: Mestre JP, Ross BH, editors. Psychology of learning and motivation. Vol. 55. Academic Press; 2011. p. 37-76.
27. VanLehn K. Structure mapping and analogical comparison in scientific thinking and learning. Educational Psychologist. 2020;55(3):143-157. https://doi.org/10.1080/00461520.2020.1773147.
28. Wang P, Patel M. Human-level AI needs the developmental approach: A position paper. Proceedings of the Seventh Annual Conference on Advances in Cognitive Systems. 2019. p. 12-27.