SIN-CHN SCIENTIFIC PRESS PTE.LTD.

AI Insights 2025, 1(1), 1973.
https://doi.org/10.59400/aii1973

Article

Resources management and execution mechanisms for thinking operating system

Ping Zhu[1,2,*], Pohua Lv[1], Weiming Zou[3], Xuetao Jiang[1], Jin Shi[3], Yang Zhang[4], Yirong Ma[3]

[1] Beijing Broad Network and Information Company Limited, Beijing 101111, China
[2] Tellhow Institute of Smart City, Beijing 100176, China
[3] Beijing Tellhow Intelligent Engineering Company Limited, Beijing 100176, China
[4] Beijing Yizhuang Smart City Institute Group Company Limited, Beijing 100176, China
*** Corresponding author:** Ping Zhu, 1401626437@qq.com

CITATION

Zhu P, Lv P, Zou W, et al. Resources management and execution mechanisms for thinking operating system. AI Insights. 2025; 1(1): 1973.
https://doi.org/10.59400/aii1973

ARTICLE INFO

Received: 17 January 2025
Accepted: 26 March 2025
Available online: 1 April 2025

Abstract: To achieve interpretable machine intelligence surpassing human cognitive levels and realize the ultimate objective of co-evolutionary human-computer interactions, this article analyzed various related aspects such as the human-computer interaction process, knowledge base construction, visual programming tool development, and thinking operating system design. This article proposed a method for simulating human thinking processes by computer: Firstly, it clarified the route by starting from the "teaching and learning" mode, which was the human-computer interaction computing mode, enabling the gradual accumulation of knowledge and data, and established the thinking knowledge base. Secondly, it established human thinking simulation mechanisms on the thinking operation system, including state perception, common sense judgment, error rollback, static logic structure analysis for the programs, and dynamic execution path analysis. Thirdly, it discussed the computer implementation methods of the thinking operation system and applications in detail, using mechanisms such as autonomous enumeration and rule induction of input data features, common sense judgment rollback, automatic error self-healing, online self-programming, and system adaptation (generalized pattern matching); all the above mechanisms were commonly used in human thinking. Finally, it summarized the whole article, and the future research directions were proposed by the authors.

Keywords: interpretable artificial intelligence; human cognition; human-computer interaction; thinking operating system; thinking simulation; thinking knowledge base

1. Introduction

Natural language semantic understanding [1–3] was an important foundation for machine thinking. The semantic expression forms of natural language are diverse and complex. In the process of analyzing the corpus, it was necessary to face practical challenges such as sparse language features [4], scattered acquisition for semantic expression forms, and long accumulation cycles for "smarter" computing models [5]. Traditional software development methods [6–8], in situations where semantic feature processing requirements could not be clearly defined, made rapid increases in the cost and complexity of processing, maintaining, and upgrading complex logic algorithms while the algorithms scaled up [9,10]. Using instances to gradually accumulate experience was the necessary way to implement the complex logic of intelligent systems [11]. To avoid frequent and large-scale logic changes in software development, it was necessary to design the software evolution model with simple

computing architecture, a machine adaptive framework, a self-learning mechanism, common sense judgment, and an abnormal self-healing ability [12–15].

The large language model based on deep learning [16–18] met the above requirements. After the neural network scale was determined, massive corpus training was carried out, and the obtained large language model could respond to external stimuli based on probability, achieving "intelligence emergence" [19–23]. However, the performances of large language models in reasoning interpretability [24–26], real-time data retrieval, implicit semantic reasoning, and numerical value inequality deduction were still at the intelligence level of "preschoolers" [27–29]. Although scholars had made improvements through technologies such as the thought chain, their performances were still unsatisfactory due to the constraints of the core technology route, and the non-interpretability of the system responses also posed safety and ethical concerns.

Traditional logic-based methods naturally had interpretability in reasoning; however, they did not have the advantages of simple architecture, machine adaptation, self-learning, commonsense judgment, and error self-healing ability as the large-scale neural computing [30–34]. Currently, logic-based methods are not the mainstream direction of general artificial intelligence; however, the authors believed that through improvements in software engineering methods, global semantic analysis, and human thinking mechanism simulation, the logic-based methods will take on the responsibility for surpassing the intelligence level of the humans and harmoniously interacting with humans to evolve together [35,36]. These improvements were the applications of software development ideas like the low-code platforms [37], perceiving natural language analysis features and processing methods, enhancing visual programming capabilities, reducing the development and maintenance costs of complex logic algorithms, etc. These methods were not mainly targeted at specific industries as the currently popular low-code platforms [38], but rather the general basic programming platform for complex logic algorithms [39–41]; Different from Microsoft C++'s MFC toolkit for GUI and the widely used Qt toolkit, the new basic programming platform could not only be applied to GUI development and maintenance but also achieve breakthroughs in general logic flow visualization, which had significant value for complex logic algorithm maintenance.

2. Related work

Since OpenAI released GPT-3.5 at the end of 2022, the application scenarios of generative artificial intelligence continued to be expanded, and large language modeling technology had become popular worldwide. In the debate on human thinking simulation between the neural network and the logic rule (symbolic intelligence) in the field of artificial intelligence, the neural network had temporarily gained a decisive advantage and become the mainstream technological direction. However, this article argued that the execution process of symbolic intelligence systems was closer to human thinking and reasoning; the execution mechanism belonged to the "white box" category, and its operation mechanism could be explained and traced in engineering; there were no safety and ethical problems, and the operation mechanism could interact with humans at any time to achieve co-evolution.

The simulation of human thinking could be divided into three steps: external input semantic understanding, comprehensive thinking mechanism implementation, and super large-scale knowledge engineering system construction [42]. Inspired by large language models' full space language input processing capabilities [43], text semantic understanding [44] could be achieved through massive clause pattern matching and global semantic inheritance and overloading techniques [45]. The comprehensive thinking mechanism realized mathematical calculations, deduction, unequal and equivalent relation deduction, proportional deduction, induction, equation resolving, etc. Among them, as an important way of creative thinking [46], the hypothesis could be proposed by induction [47], which could be incorporated into the knowledge base [48] for unified management and access after it was verified by subsequent samples. In restricted domains, thinking actions included but were not limited to mathematic actions such as addition, subtraction, multiplication, division, set building, number axis analysis, coordinate analysis, accumulation, counting, as well as process control actions such as backtracking, looping, recursion, enumeration, equation resolving, assumptions, induction, etc. Together, they were converted into an industry application framework like the low-code platform [49], with the ability to fully integrate domain thinking actions. In general, all thinking processes could be completed by combining the elements of the thinking action set, and some temporarily requiring highly intelligent thinking actions could be obtained by human-computer interaction [50] and negotiation. In this article, the functions were encapsulated by the frameworks, which were more like the function "shell" with contexts. Due to thinking actions could also be encapsulated by frameworks, the framework could provide data records of action (function) inputs, outputs, and states. The system could achieve visual programming [51] capability by this encapsulation approach [52]. Unlike popular visual programming methods such as Microsoft MFC and Qt components, this article used software static structure and dynamic execution path tracking to help programmers understand the workflow of the software system. The thinking operating system [53] could be constructed for machines, where all thinking actions were scheduled, executed, monitored, maintained, upgraded, rolled back, or abandoned on this platform, forming the support platform (or integrated development environment) for software development, debugging, and maintenance of complex intelligent applications.

This integrated development environment [54] could be used to develop complex logic algorithms with "automatic", "self-healing", "online", and "self-programming" features. "Automatic" referred to the intelligence and autonomy of the system's analysis and processing, "self-healing" referred to the system's error self-recovery, "online" referred to the system updating without interrupting software execution, and "self-programming" referred to the ability to plan, predict, execute, verify, and rollback action sequences. From the current research progress at home and abroad, necessary human-computer interaction [55] was one of the important capabilities for interpretable systems to achieve human-machine co-evolution and improvement. For example, there were various types of data input modes, and the pattern reference function based on instance analysis often could not select the optimal thinking action due to sparse input data pattern features; On the other hand, the generalization method [56] of the input data pattern by the system was another important ability that the

thinking operating system must possess; all of these required the system to have the ability to propose inductive hypotheses with a small number of use cases and then verify and confirm the hypotheses through human-computer interaction. To solve problems based on understanding, planning, thinking, and action sequences, the systematic semantic feature vocabulary system that represented action semantics was needed. Building such a feature vocabulary system was an extremely difficult task, and there was currently no related work that could be retrieved in this area. It could only rely on the author's accumulated experience in humanoid machines resolving elementary mathematic application problems [57–60], continuously processed and accumulated examples, gradually improved and upgraded, and ultimately achieved this design target.

3. System architecture

The system architecture design is based on the five assumptions as follows:

(1) The basic data structures have been determined, and appropriate expansion slots have been reserved previously.
(2) The quantity of all language feature recognition functions with the basic data structures is limited.
(3) The system workflows are all composed of known thinking actions.
(4) The quantities of other features, such as static data structure features, and dynamic action history features are also limited.
(5) In addition to a few thinking actions that require human experts to add to the system interactively, whether modifying data structures or performing other thinking actions, are all planned, executed, and validated by machines automatically (as shown in **Figure 1**).

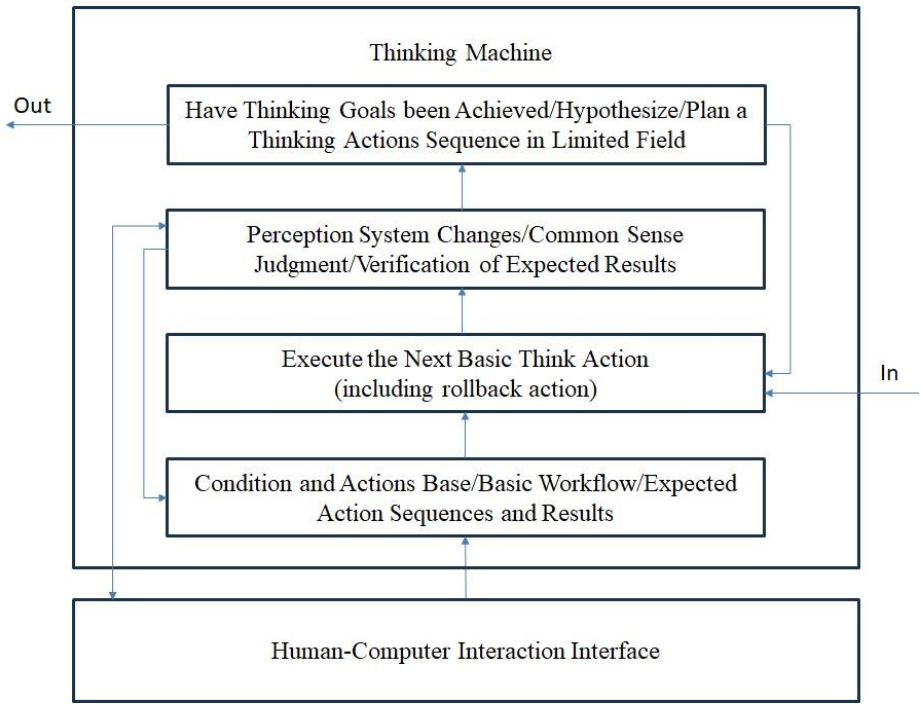

Figure 1. Computing model.

3.1. Computing model

At present, thinking machines are still in the research and development stage with the "human teaching and machine learning" mode, and the autonomous interaction and self-learning for thinking machines are still laid in the initial conceptual demonstration process. All thinking processes are based on a defined sequence of thinking actions (condition modules and action modules). Thinking action sequence, which can display the static logic structure, dynamic execution sequence, and system state in a limited field, runs on top of the thinking operation system, as well as rollback and traversal. The machine learning process is also based on the thinking operating system, and the learning and supervision processes are also completed by pre-set thinking actions (except for a few new action modules).

Thinking machines required humans to initialize the conditions, actions, and basic process sets, as well as the operation mechanism of the thinking operating system to maintain and call these preset modules and processes. In the process of problem-solving, the expected solution, steps, and results of the problem could also be used as verification information for interactive input into the thinking operating system. When the problem could not be solved correctly, the preset modules and processes could also be modified or supplemented through the human-computer interaction interface.

3.2. Thinking workflows

The basic workflows of the thinking machine were as follows: firstly, it performed the analysis and processing action based on preset modules and actions, then perceived the state changes of the thinking machine, judged the step results based on common sense, and determined whether the stage nodes had achieved the expected goals. Based on these thinking mechanisms, the preset knowledge base could be upgraded, and human intervention could be sought through a human-computer interaction interface. It was also possible to check whether the problem had been solved, propose new hypotheses, plan the next thinking action, and then loop through the next choice thinking action. Until the problem was solved and the results and thinking steps of the solution were output. Among them, system state perception included the recognition of preset knowledge features/attributes, input data information features/attributes, and the action-executing sequence.

The concept of the thinking operating system was based on the premise of pre-set modules, supplemented by human-computer interaction. Only in this way can we achieve the goals of machines autonomously, independently thinking, and self-healing from errors. Of course, these goals could be gradually achieved through human-computer interaction.

3.3. Thinking actions executing environment

Similar to "toddler", which always inherited the genes of their parents, the basic framework of self-supervised learning for thinking machines was determined by the accumulation and summary of previous case analysis experience. During the thinking process, the system perceived the features of data processing, logic functions, and workflow (static/dynamic), which constructed language feature knowledge pattern hypotheses based on these features. After machine cross-validation, self-supervised

validation, new knowledge was formed. This progressive supervised learning method with an initial knowledge set might require human experts to add data structure elements, feature recognition functions, and action workflows to form an action execution environment constrained by limited operating space in the thinking machine operating system (as shown in **Figure 2**).

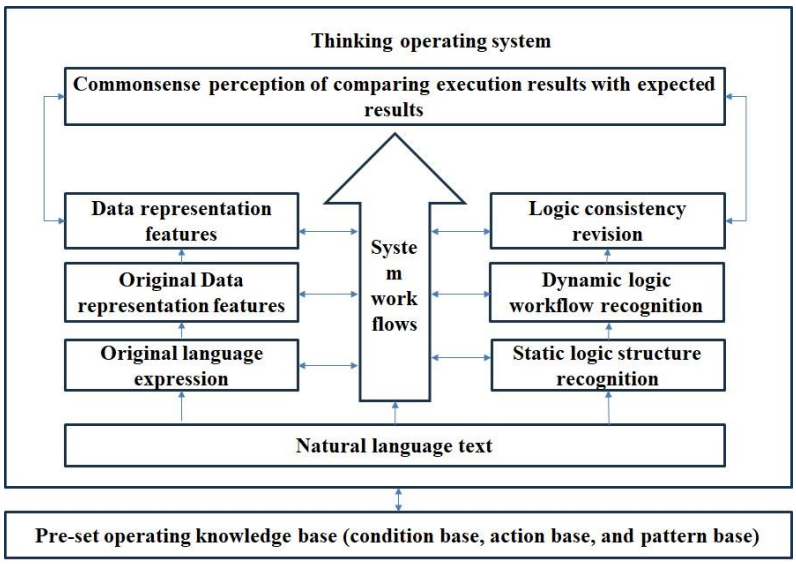

Figure 2. Thinking actions executing environment.

The preset thinking knowledge base usually included thinking auxiliary knowledge such as action execution conditions, action bodies, and various thinking modes, which provided an initial framework plan for the thinking action execution. In the actual running process of the thinking operation system, it was continuously supplemented, improved, abandoned, and backtracked. With a dynamic queue structure that connected the beginning and ending, the action execution plan was proposed and verified step by step, achieving the iterative loop of continuous thinking without termination until the problem was solved or the termination execution conditions were met. The thinking operating system realized the scheduling and matching of various knowledge in the preset thinking knowledge base and planned for the execution and verification of operation actions step by step. After natural language text input, the thinking operating system perceived the features of the original input data, the representation features of intermediate results, as well as the original static logic processing workflow structure and dynamic processing action sequence, and reflected, evaluated, verified, hypothesized, or backtracked on per action execution step. The consistency recognition and checking mechanism of the original static logic processing workflow structure, as well as the perception, evaluation, probing, or rollback mechanism of the dynamic execution action sequence, were the core functions of the thinking operating system.

This section divided the problem-solving process into interpretable basic thinking actions and execution environment, where the execution environment planned, called, monitored, judged, and rolled back basic thinking actions, as well as explained how to design the execution environment.

4. Core functions

The thinking operating system integrated the commonsense knowledge base, domain knowledge base, action condition base, and thinking action base, and had the abilities to schedule, execute, monitor system status, evaluate results, and roll back thinking actions of the problem-resolving process. On top of the thinking operating system, the intelligent module with a closed-loop action queue could be built, which had the ability to dynamically plan thinking actions and provided the basic development environment for machine thinking's autonomous, automatic self-healing, and online self-programming capabilities.

4.1. Basic concepts

a) Variable attributes

To simplify and display the data flow as much as possible, this article suggested minimizing the use of local variables (stack) and maximizing the use of global variables (heap) for data transfer across functions. The optimization and compression for storage space occupied by the software global variables could be achieved with specific tools.

b) Node types

To explicitly control and display logic functions, the program was represented as the network composed of function nodes. All nodes were composed of Boolean-type nonparametric functions; the internode connection represented the flow direction of the execution location. Usually, node functions could include four basic types, such as general nodes, loop (head) nodes, loop (tail) nodes, and recursive nodes.

c) Rollback mechanism

All function nodes had the rollback function, which restored the changes made by this function node to global variables. Combining a rollback mechanism with common sense judgments could achieve dynamic programming of logic processes.

d) Function pattern

The function pattern, composed of condition nodes and function node sequences, was used for the automatic intelligent construction of software macro workflow. Data attribute features and co-relationships could be integrated into the function patterns.

e) Pattern data attributes and associated features

Data attributes and associated features were mainly used for the implementation condition nodes and logic judgment. Programs could usually be considered as consisting of data flows and workflows. The node sequence pattern was composed of the data flows and the workflow pattern, and the pure data flows represented the data processing and data transmission of function nodes.

For example, the function nodes could be represented by the framework using the following C-like language data structure:

```
struct STRUCT_FUNTION_ENTRANCE
{
Boolean (*App)();
///Function pointer
Char str_annotation [MAX_ATTRIBUTES_LIST_LEN]; /// The text string describing what the current function could do.
```

Boolean bRet;///Function returned the Boolean value.

};

struct STRUCT_GENERAL_FRAME_INTERFACE

{

Long nID;/// Current framework node identifier ID;

Boolean bInit=false;/// "false" for the first execution of the loop node, "true" for all others.

Int nType;///Divided into recursive nodes, loop head nodes, loop tail nodes, general nodes, etc;

*struct STRUCT_GENERAL_FRAME_INTERFACE * pBrother;///Pointed to the brother framework node.*

*struct STRUCT_GENERAL_FRAME_INTERFACE * pStep;///Pointed to the next deduction step framework node.*

*struct STRUCT_GENERAL_FRAME_INTERFACE * pTNext;///Pointed to the next framework node when the function returned "true".*

*struct STRUCT_GENERAL_FRAME_INTERFACE * pFNext;///Pointed to the next framework node when the function returned "fslse".*

*struct STRUCT_GENERAL_FRAME_INTERFACE * pSon;///Pointed to the first son framework node.*

*struct STRUCT_GENERAL_FRAME_INTERFACE * pFather;///Pointed to the parent framework node.*

*struct STRUCT_FUNTION_ENTRANCE * fun_init;/// Initialized the logic branch variables.*

*struct STRUCT-FUNTION-INTRANCE * fun_execute_body;///Logic branch function body.*

*struct STRUCT-FUNTION-INTRANCE * fun_rtn;///Logic branch function body returned values passing function.*

*Boolean (*is_result_ validated) ();*

///Function pointer for result validation.

*Boolean (*Record_Input_Data) ();*

///Function pointer for input data recording function.

*Boolean (*Record_Output_Data) ();*

///Function pointer for recording output results.

*Boolean (*Record_Output_Format_Return) ();*

///Function pointer for unified output format function.

};

4.2. Structure comparison algorithm

The like-C language Algorithm 1 could be represented as follows:

Algorithm 1 Structure comparison algorithm

Input: New module function feature string FString, new module input data feature string DString; The static logic structure web of the new module MWeb, the static logic structure web of the software SWeb and current node pointer pNode; the features base of software nodes FBase; Feature vocabularies base FVBase.
Output: The module locations of different static logic structures MLocset.
Algorithm description: structure_comparison_algorithm().
(1) pNode <= SWeb; /// compare from the head node of SWeb.
(2) IF (!is_similar_modules(MWeb, FString, DString, pNode, SWeb, FBase, FVBase)) THEN RETURE FALSE; /// if new module node has not similar node in SWeb, return failure.
(3) pNode <= pNode->Son; /// current node changes to its Son node.
(4) IF (!is_similar_son_nodes(MWeb, FString, DString, pNode, SWeb, FBase, FVBase)) THEN RETURE FALSE; /// if new module node has not similar Son node with current location of SWeb, return failure.
(5) pNode <= pNode->TNext; /// current node changes to its TNext node.
(6) IF (!is_similar_tnext_nodes(MWeb, FString, DString, pNode, SWeb, FBase, FVBase)) THEN RETURE FALSE; /// if new module node has not similar TNext node with current location of SWeb, return failure.
(7) pNode <= pNode->FNext; /// current node changes to its FNext node.
(8) IF (!is_similar_fnext_nodes(MWeb, FString, DString, pNode, SWeb, FBase, FVBase)) THEN RETURE FALSE; /// if new module node has not similar FNext node with current location of SWeb, return failure.

4.3. Nodes scheduling algorithm

Each node *G* function part could be composed of the initialization part *G.Init*, the body part *G.Body*, and the return part *G.Retn*. If node G contained the child node *I*, then the execution sequence of node *G* was *G.Init* → *I.Init* → *I.Body*→ *I.Retn* → *G.Retn*; the *G.Body* actually did not execute. The like-C language Algorithm 2 could be represented as follows:

Algorithm 2 Nodes scheduling algorithm

```
Input: The static logic structure web of the software SWeb and current node pointer pNode.
Output: The problem solution, middle steps and middle results.
Algorithm description: nodes_scheduling_algorithm().
(1) assign_nodes_depth(pNode, SWeb) /// assigns the Son relation depth for each nodes.
(2) IF(is_current_general_node(pNode)) THEN
    IF(!is_has_son(pNode))THEN
        pNode->Init(); bRet = pNode->Body(); pNode->Retn();
        IF(bRet && is_has_TNext(pNode)) THEN
            align_return_depth();
            pNode <= pNode->TNext;
            GOTO (2);
        IF((!bRet)&&( is_has_FNext(pNode))) THEN
            align_return_depth();
            pNode <= pNode->FNext;
            GOTO (2);
    ELSE /// is_has_son(pNode)
        pNode->Init(); pNode <= pNode->Son;
(3) IF(is_current_embed_node(pNode)) THEN
    IF(!is_has_son(pNode))THEN
        IF(b_first_run_flag(pNode)) THEN
            pNode->Init();
        bRet = pNode->Body();
        IF(bRet && is_has_TNext(pNode)) THEN
            align_return_depth();
            pNode <= pNode->TNext;/// GOTO (3);
            GOTO (2);
        IF((!bRet)&&( is_has_FNext(pNode))) THEN
            b_first_run_flag(pNode) <= FALSE;
            align_return_depth();
```

Algorithm 2 (*Continued*)

```
30:                         pNode <= pNode->FNext;
31:                         GOTO (2);
32:     ELSE    /// is_has_son(pNode)
33:                 pNode->Init(); pNode <= pNode->Son;
34: (4) IF(is_current_lpbn_node(pNode)) THEN/// loop head node
35:     IF(b_first_run_flag(pNode)) THEN
36:                  pNode->Init();
37:     bRet = pNode->Body();
38:     IF(bRet && is_has_TNext(pNode)) THEN
39:                 align_return_depth();
40:                 pNode <= pNode->TNext;/// GOTO (4);
41:                 GOTO (2);
42:     IF((!bRet)&&( is_has_FNext(pNode))) THEN
43:                 b_first_run_flag(pNode) <= FALSE;
44:                 pNode <= pNode->FNext;
45:                 GOTO (2);
46: (5) IF(is_current_lptl_node(pNode)) THEN /// loop tail node
47:     pNode->Init(); pNode->Body(); pNode->Retn();
48:     GOTO (4);
```

4.4. Action planning algorithm

All thinking actions (condition nodes, function nodes, recursive nodes, loop (head) nodes, and loop (tail) nodes) and their sequence fragments were evaluated step by step at the current time, and the optimal action sequence was selected to execute. Machine implementation for the progressive program logic (node sequence) determination was very difficult. Therefore, this article used hook function and callback function mechanisms to pre-set templates and later determined the method to refine the planning of thinking action sequences. The like-C language Algorithm 3 could be represented as follows:

Algorithm 3 Action planning algorithm

```
1: Input: thinking actions set ActionSet, candidate thinking actions queue CandActionQueue.
2: Output: candidate thinking action T, current system state S, problem results.
3: Algorithm description: actions_planning_algorithm().
4: (1) CandActionQueue <= is_can_activate(ActionSet, S) ///push the thinking actions that can be activated in current
   state into CandActionQueue.
5: (2) T <= select_optimal_action(CandActionQueue) /// select the optimal thinking action in current system state.
6: (3) S' <= exec_action(T); /// execute the selected action, record the output data to new system state.
7: (4) IF (is_problem_solved(S')) THEN RETURN TRUE; /// if problem was solved, then return successfully.
8: (5) ELSE S <= S'; GOTO (1). /// if problem was not solved, update system state, jump to step (1).
```

4.5. State perception objects

State perception objects included the input data, middle result structure, and attribute features; common sense judgment for thinking action results; historical data features and the trends for the results; current system state features; thinking modes; human-machine interaction for verifying results; and rules induction for the input data, etc.

4.6. Thinking knowledge base

The thinking operating system was built on the aggregation of a large amount of commonsense knowledge and commonsense features. The thinking knowledge base

could usually contain various static data and formal semantic features of thinking functions, as well as dynamic historical evolution features and semantic features of data and actions. The knowledge base provided all the basic thinking actions and mechanisms for the thinking operating system. Except for a small number of creative thinking scenes that required human-computer interaction (which would become less and less necessary as the knowledge base and meta-knowledge converge), other main thinking functions could achieve autonomous closed-loop operation through pattern matching or basic working mode combination. At present, the knowledge base is still designed, built, and maintained entirely manually. The semi-automatic construction system with manual assistance and the automatic construction system with self-inducing hypothesis verification are already in the requirements demonstration stage. However, due to the complexity of this issue and the involvement of human-machine interaction dynamic updates, there are currently no specific techniques that contribute to the industry.

This section summarized the basic concepts of the thinking operating system, introduced the execution environment of thinking actions based on the function pointer framework, explained the core functions of the thinking operating system using several algorithms, the static data feature perception, and the construct of the thinking knowledge base.

5. Key application technologies

5.1. Autonomous enumerating input data features and inducting action laws

It is impossible to exhaustively enumerate data patterns, so it is necessary to generalize patterns, as well as extend the ability to process patterns. For example, the initial input vocabulary sequence "...S1...R1...", where S1 was the source phrase (vocabulary sequence fragment), R1 was the referential phrase (vocabulary sequence fragment), and the action should replace the referential phrase with the source phrase to obtain the vocabulary sequence "...S1...S1...". In order to generalize the processing ability of language referential phenomena, the thinking operating systems needed to have the ability to autonomously enumerate and interact with humans to obtain the following extended input vocabulary sequences and their correct results for referential resolving:

"....." (none referential phenomenon; there was no matched source phrase and reference phrase)

"...S1..." (none referential phenomenon, but matching source phrase)

"...R1..." (none referential phenomenon, but matching referential phrase)

"...S1...R1...R1..." (There were referential phenomena, where two referential phrases should be replaced by the same source phrase)

"...S1...S2...R2..." (There was a referential phenomenon; the reference phrase should be replaced by the adjacent source phrase that appeared earlier)

"...S1...R1...S2...R2..." (There were referential phenomena, where both reference phrases should be replaced by their adjacent source phrases that appeared earlier)

…

Sometimes, humans also could not master the real laws with limited input patterns. Thus, the thinking operating system could enumerate input patterns and inquire with humans for process actions, then retrieve the action base and generalize action sequences based on reference features, such as replacing with the first appearing source phrase, replacing with the forward adjacent source phrase, or replacing with the last appearing source phrase. Finally, new knowledge was obtained through human-machine interaction learning; that is, the vocabulary pattern reference phenomena could be resolved by forward adjacent vocabulary phrase substitutions.

5.2. Rollback action

Every thinking action's result must be verified and judged by common sense knowledge. For example, the numbers of people, trees, and machines only have integer values, and if the derived results were decimals, then it could be concluded that there were errors. The operating system should activate the rollback mechanism to rethink. There was still a lot of common sense like this, such as, within the scope of elementary mathematics, all data element variables could not be assigned negative values; the divisor could not be zero; the airplane speed was usually faster than a train and a car. The accumulation and access of common sense were both implemented by the thinking operating system. The results obtained from human-computer interaction could also be supervised learning through common sense rollback. Variables without initial values could not participate in calculations, or empty address structure pointer's elements could not be accessed during program execution. They were also common knowledge to activate rollback action.

The thinking action function nodes were all designed as function pointer frameworks (including initialization module pointer, function body pointer, and return module pointers), with clear logic meaning for output results. The thinking operating system could add a "callback function" to the return function to execute a commonsense judgment function. Because the thinking action call was designed as a deep traversal chain with historical records, the chain has father node pointers and brother node pointers, so the "error rollback" could be achieved. For example, there were function nodes 1–7; the current software execution workflow was illustrated as follows in **Figure 3**:

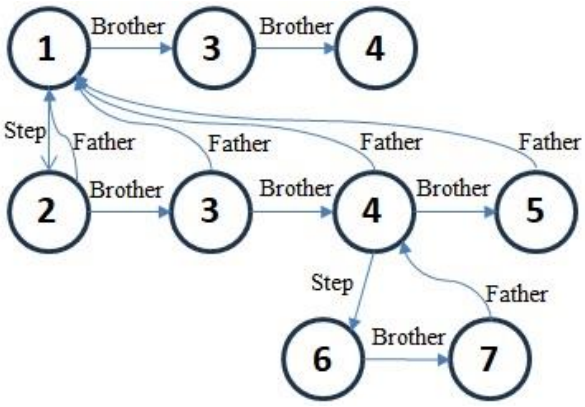

Figure 3. Dynamic node execution workflow.

At the initial stage, function nodes 1, 3, and 4 satisfied the action activation conditions and were linked by brother pointers; after function node 1 was executed, function nodes 2, 3, 4, and 5 satisfied the action activation conditions and were also linked by brother pointers. When function node 2 failed, execute function node 3 according to the brother pointer of function node 2. If a function node failed and its brother pointer was NULL, then execute the brother node of its father node.

5.3. Error self-healing

At each step of the thinking process, the operating system recorded all possible reasoning and computing actions, selected the most likely appropriate action to execute, and if commonsense judgment was abnormal at this time, it took the rollback action and selected the next appropriate action to execute... If all possible actions that could be executed by the previous level node failed, it continued to roll back according to the above idea until the system obtained the right answer or all possible action paths failed. The self-healing implementation mechanism was similar to error rollback and would not be elaborated on in this article.

5.4. Online self-programming

The thinking actions were virtually served in the form of callback functions, and the function entities were dynamically called in the form of dynamic link libraries (DLL). All predicted access function entities were queued head-to-end. Dynamically accessing function entities during the execution of the main function enabled online access to unknown function entities that satisfied callback function parameters. The continuous evolution of thinking could be realized in this closed-loop queue.

5.5. Generalized pattern matching

The processing capability of thinking operating systems was generalized to cover all language expression patterns by templates (attributes pattern matching such as data elements, parts of speech, person names, place names, institution names, association patterns, commonsense knowledge reasoning, as well as thinking modes such as exploration, enumeration, hypothesis, prediction, deduction, and induction, etc.). The coverage of all executable actions by pre-set bases and the resolution of synonyms and referents were also important ways of language expression generalization. The trend deduction of pattern matching was a generalized way of enumerating thinking actions.

5.6. Experiment analysis

The demo system had been developed for 5 years, and more than 1100 math application problems of different difficulty levels were debugged, such as general math problems and Olympic math problems. The system knowledge base stored pre annotated 2920 clause semantic frameworks, 352 global semantic frameworks, 153 mathematic knowledge item, 180 general mathematic attribute items, 67 commonly used formulas, and 134 resolving rules, as well as preset data such as dictionaries, concept membership relationship knowledge graphs, concept attribute relationship knowledge graphs, scene synonym lists, concept reference patterns and scene reference patterns, segmentation vocabulary sequence correction and segmentation

pattern correction relationship tables, part of speech vocabulary sequence correction and part of speech pattern correction relationship tables, problem semantic component patterns and corresponding variable retrieval patterns, equivalence association patterns, logic thinking commonsense databases, feature constraint tables, semantic component meta frameworks and corresponding meta framework retrieval patterns, named entity examples, time representation examples, etc. The semantic annotation and debugging of general math problems took an average of 1–3 h; the semantic annotation and debugging of Olympiad math problems took an average of 2–3 days. The number of general calculation formulas and conditional formulas in the general math problem release mode of the demo system was generally less than 100, and the average executing time for single machine resolving was 20–30 min. The number of general calculation formulas and conditional formulas in the Olympic math problem release mode was generally between hundreds and thousands, and the average executing time for single machine resolving was 24–36 h. At present, the logic flows of the demo system are very complex; the efficiencies of debugging and executing are far lower than the requirements of engineering development, so it is urgent to use the thinking operating system for standardized development, debugging, and maintenance.

This section explained several key application technologies such as feature enumeration, action rollback, error self-healing, online programming, and pattern generalization, as well as an experiment analysis of the demo system.

6. Conclusion

Although scholars had proposed some meaningful human cognitive architectures internationally, e.g., ACT-R [61,62] and SOAR [63], the implementations for specific engineering problems were still not satisfactory. This article had developed a prototype of the interpretable thinking operation system based on semantic pattern matching and a finite set of thinking actions (including basic thinking actions such as addition, subtraction, multiplication, division, taking absolute values, calculating area, calculating perimeter, rollback, etc.). It could display the semantic understanding and analyzing processes, software static module structure, and software dynamic execution process and could serve as the basic platform for in-depth research. The difficulty in implementing the thinking operating system lay in the gradual acquisition of sufficient think action samples through human-computer interaction, as well as the activation and result verification mechanisms of thinking patterns such as backtracking, probing, enumeration, prediction, and hypothesis. This article elaborated on the key factors and solutions faced by the development of thinking operating systems from different perspectives.

There are many research directions for the expansion of this article. In the future, based on this prototype, we would research and develop an online continuous learning system for commonsense based on Internet information and build a series of special knowledge bases for machine thinking (knowledge maps such as concept membership, coordinating relationship, attribute relationship, etc.); We would perform mathematical, conceptual, or logical induction on the information in the initial knowledge base, propose hypotheses, validate, and update the knowledge base through

continuous learning; we also planned to explore the mechanism of deep thinking (including the derivation of mathematical symbol systems) and upgrade the thinking control mechanism in the prototype; we would showcase this new programming approach for complex logic algorithms by designing and developing the independent action planning engine and the commonsense knowledge bases. Due to the small scale of the demo system data and the successful validation of the core algorithms a moment ago, this article condensed the common functions into the operating system, with the main design goal of reducing the complexity of system logic updating and the convenience of system maintenance. Maintaining efficient system operation in large-scale data and complex tasks faced by future application systems was discussed in another article [64].

Author contributions: Conceptualization, methodology, software, writing—original draft, PZ; project administration, resources, PL; formal analysis, supervision, WZ; investigation, XJ; data curation, JS; writing—review and editing, YZ; validation, YM. All authors have read and agreed to the published version of the manuscript.

Conflict of interest: The authors declare no conflict of interest.

References

1. Wang Y. Research and Application of Machine Learning Algorithm in Natural Language Processing and Semantic Understanding. In: Proceedings of the 2024 International Conference on Telecommunications and Power Electronics (TELEPE); 2024. doi: 10.1109/telepe64216.2024.00123
2. Khummongkol R, Yokota M. Proposal of Human-like Spatiotemporal Language Understanding Based on Mental Image Model for Language-centered Human-Robot Interaction. In: Proceedings of the 2023 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunications Engineering (ECTI DAMT & NCON); 2023. doi: 10.1109/ectidamtncon57770.2023.10139743
3. Goswami K, Dutta S, Assem H. Mufin: Enriching Semantic Understanding of Sentence Embedding using Dual Tune Framework. In: Proceedings of the 2021 IEEE International Conference on Big Data (Big Data); 2021. doi: 10.1109/bigdata52589.2021.9671614
4. Arisoy E, Saraclar M, Roark B, et al. Syntactic and sub-lexical features for Turkish discriminative language models. In: Proceedings of the 2010 IEEE International Conference on Acoustics, Speech and Signal Processing; 2010. doi: 10.1109/icassp.2010.5495226
5. Sun L, Yan H. Feature Fusion Transformer Network for Natural Language Inference. In: Proceedings of the 2022 IEEE International Conference on Mechatronics and Automation (ICMA); 2022. doi: 10.1109/icma54519.2022.9856400
6. Kuhrmann M, Tell P, Hebig R, et al. What Makes Agile Software Development Agile?. IEEE Transactions on Software Engineering. 2022; 48(9): 3523-3539. doi: 10.1109/tse.2021.3099532
7. Lontsikh PA, Gulov AE, Livshitz II, et al. System-oriented Analysis and Classification of Process Control Methods for Software Development. In: Proceedings of the 2021 International Conference on Quality Management, Transport and Information Security, Information Technologies (IT&QM&IS); 2021. doi: 10.1109/itqmis53292.2021.9642850
8. Faruk MJH, Subramanian S, Shahriar H, et al. Software Engineering Process and Methodology in Blockchain-Oriented Software Development: A Systematic Study. In: Proceedings of the 2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA); 2022. doi: 10.1109/sera54885.2022.9806817
9. Kim J, Kim J, Jo H, et al. Multiple Domains Knowledge Graph Search via Heuristic Algorithm for Answering Complex Questions. In: Proceedings of the 2021 10th International Congress on Advanced Applied Informatics (IIAI-AAI); 2021. doi: 10.1109/iiai-aai53430.2021.00080
10. Gou X, Zhou P, Yang H, et al. Identifying Influential Nodes in Complex System from Multiscale Complex Network Perspective. In: Proceedings of the 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security Companion (QRS-C); 2023. doi: 10.1109/qrs-c60940.2023.00051

11. Rique T, Dantas E, Perkusich M, et al. Empirically Derived Use Cases for Software Analytics. In: Proceedings of the 2022 International Conference on Software, Telecommunications and Computer Networks (SoftCOM); 2022. doi: 10.23919/softcom55329.2022.9911514
12. Shadab N, Cody T, Salado A, et al. Closed Systems Paradigm for Intelligent Systems. In: Proceedings of the 2022 IEEE International Systems Conference (SysCon); 2022. doi: 10.1109/syscon53536.2022.9773829
13. Yuan L, Li T, Tong S, et al. Broad Learning System Approximation-Based Adaptive Optimal Control for Unknown Discrete-Time Nonlinear Systems. In: Proceedings of the IEEE Transactions on Systems, Man, and Cybernetics: Systems; 2022. doi: 10.1109/tsmc.2021.3113357
14. Tendle A, Little A, Scott S, et al. Self-Supervised Learning in the Twilight of Noisy Real-World Datasets. In: Proceedings of the 2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA); 2022. doi: 10.1109/icmla55696.2022.00074
15. Kimura M, Pereira LK, Kobayashi I. Effective Masked Language Modeling for Temporal Commonsense Reasoning. In: Proceedings of the 2022 Joint 12th International Conference on Soft Computing and Intelligent Systems and 23rd International Symposium on Advanced Intelligent Systems (SCIS&ISIS); 2022. doi: 10.1109/scisisis55246.2022.10002012
16. Khlaisamniang P, Khomduean P, Saetan K, et al. Generative AI for Self-Healing Systems. In: Proceedings of the 2023 18th International Joint Symposium on Artificial Intelligence and Natural Language Processing (iSAI-NLP); 2023. doi: 10.1109/isai-nlp60301.2023.10354608
17. Iman M, Rasheed K, Arabnia HR. EXPANSE, A Continual Deep Learning System; Research Proposal. In: Proceedings of the 2021 International Conference on Computational Science and Computational Intelligence (CSCI); 2021. doi: 10.1109/csci54926.2021.00103
18. Msayi M, Salamntu LTP. Understanding the Benefits of Deep Learning in Drug Discovery: A Scoping Review. In: Proceedings of the 2024 Conference on Information Communications Technology and Society (ICTAS); 2024. doi: 10.1109/ictas59620.2024.10507136
19. Kapoor D, Gupta D, Uppal M. A Scientometric Review of Machine Learning and Deep Learning Techniques. In: Proceedings of the 2024 7th International Conference on Circuit Power and Computing Technologies (ICCPCT); 2024. doi: 10.1109/iccpct61902.2024.10673003
20. Örpek Z, Tural B, Destan Z. The Language Model Revolution: LLM and SLM Analysis. 2024 8th International Artificial Intelligence and Data Processing Symposium (IDAP). 2024; 3: 1-4. doi: 10.1109/idap64064.2024.10710677
21. Arulmohan S, Meurs MJ, Mosser S. Extracting Domain Models from Textual Requirements in the Era of Large Language Models. In: Proceedings of the 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C); 2023. doi: 10.1109/models-c59198.2023.00096
22. Kobayashi A, Yamaguchi S. Extraction of Subjective Information from Large Language Models. In: Proceedings of the 2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC); 2024. doi: 10.1109/compsac61105.2024.00253
23. Amin MM, Cambria E, Schuller BW. Will Affective Computing Emerge From Foundation Models and General Artificial Intelligence? A First Evaluation of ChatGPT. IEEE Intelligent Systems. 2023; 38(2): 15-23. doi: 10.1109/mis.2023.3254179
24. Kaswan KS, Dhatterwal JS, Batra R, et al. ChatGPT: A Comprehensive Review of a Large Language Model. In: Proceedings of the 2023 International Conference on Communication, Security and Artificial Intelligence (ICCSAI); 2023. doi: 10.1109/iccsai59793.2023.10421090
25. Banimelhem O, Al-khateeb B. Explainable Artificial Intelligence in Drones: A Brief Review. In: Proceedings of the 2023 14th International Conference on Information and Communication Systems (ICICS); 2023. doi: 10.1109/icics60529.2023.10330543
26. Valina L, Teixeira B, Reis A, et al. Explainable Artificial Intelligence for Deep Synthetic Data Generation Models. In: Proceedings of the 2024 IEEE Conference on Artificial Intelligence (CAI); 2024. doi: 10.1109/cai59869.2024.00109
27. Shevskaya NV. Explainable Artificial Intelligence Approaches: Challenges and Perspectives. In: Proceedings of the 2021 International Conference on Quality Management, Transport and Information Security, Information Technologies (IT&QM&IS); 2021. doi: 10.1109/itqmis53292.2021.9642869
28. Zhou L, Rao G. Chinese Stylistic Competence: Evaluation Method and Datasets of Large Language Model's Performance. 2023 International Conference on Asian Language Processing (IALP). Published online November 18, 2023: 271-277. doi: 10.1109/ialp61005.2023.10337306

29. Tong S, Liu Z, Zhai Y, et al. Eyes Wide Shut? Exploring the Visual Shortcomings of Multimodal LLMs. In: Proceedings of the 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2024. doi: 10.1109/cvpr52733.2024.00914
30. Zhong S, Huang Z, Gao S, et al. Let's Think Outside the Box: Exploring Leap-of-Thought in Large Language Models with Creative Humor Generation. In: Proceedings of the 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2024. doi: 10.1109/cvpr52733.2024.01258
31. Hang CN, Wei TC, Yu PD. MCQGen: A Large Language Model-Driven MCQ Generator for Personalized Learning. IEEE Access. 2024; 12: 102261-102273. doi: 10.1109/access.2024.3420709
32. Zhang H, Wang L, Shao H, et al. Large Model Fine-Tuning Method Based on Pre-Cognitive Inductive Reasoning—PCIR. In: Proceedings of the 2024 5th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT); 2024. doi: 10.1109/ainit61980.2024.10581497
33. Sasaki R. AI and Security—What Changes with Generative AI. In: Proceedings of the 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security Companion (QRS-C); 2023. doi: 10.1109/qrs-c60940.2023.00043
34. Huang T, You L, Cai N, et al. Large Language Model Firewall for AIGC Protection with Intelligent Detection Policy. In: Proceedings of the 2024 2nd International Conference On Mobile Internet, Cloud Computing and Information Security (MICCIS); 2024. doi: 10.1109/miccis63508.2024.00047
35. Liang BS. AI Computing in Large-Scale Era: Pre-trillion-scale Neural Network Models and Exa-scale Supercomputing. In: Proceedings of the 2023 International VLSI Symposium on Technology, Systems and Applications (VLSI-TSA/VLSI-DAT); 2023. doi: 10.1109/vlsi-tsa/vlsi-dat57221.2023.10134466
36. Tao Y, Wu C, Li J, et al. Research on Frontier Issues of New Generation Artificial Intelligence. In: Proceedings of the 2020 IEEE 3rd International Conference on Electronic Information and Communication Technology (ICEICT); 2020. doi: 10.1109/iceict51264.2020.9334257
37. Nazar M, Alam MM, Yafi E, et al. A Systematic Review of Human–Computer Interaction and Explainable Artificial Intelligence in Healthcare With Artificial Intelligence Techniques. IEEE Access. 2021; 9: 153316-153348. doi: 10.1109/access.2021.3127881
38. Phalake V, Joshi S, Rade K, et al. Modernized Application Development Using Optimized Low Code Platform. In: Proceedings of the 2022 2nd Asian Conference on Innovation in Technology (ASIANCON); 2022. doi: 10.1109/asiancon55314.2022.9908726
39. Wen X, Hu L. The research of C language programming intelligent scoring technology based on the semantic similarity. In: Proceedings of the 2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet); 2012. doi: 10.1109/cecnet.2012.6202278
40. Jafari SM, Yildirim S, Cevik M, et al. Anaphoric Ambiguity Resolution in Software Requirement Texts. In: Proceedings of the 2023 IEEE International Conference on Big Data (BigData); 2023. doi: 10.1109/bigdata59044.2023.10386192
41. Ni B, Lo LY, Leung KS. A generalized sequence pattern matching algorithm using complementary dual-seeding. In: Proceedings of 2010 IEEE International Conference on Bioinformatics and Biomedicine (BIBM); 2010.
42. Zhu P. Thinking Machine. Nova science publisher; 2024. doi: 10.52305/bqgy1221
43. Jayakody R, Dias G. Performance of Recent Large Language Models for a Low-Resourced Language. In: Proceedings of the 2024 International Conference on Asian Language Processing (IALP); 2024. doi: 10.1109/ialp63756.2024.10661169
44. Alam YS. Lexical-semantic representation of the lexicon for word sense disambiguation and text understanding. In: Proceedings of 2009 IEEE International Conference on Semantic Computing; 2009.
45. Zhu P, Lv P, Shi J, et al. Semantic Inheritance and Overloading. In: Proceedings of the 2022 IEEE 2nd International Conference on Software Engineering and Artificial Intelligence (SEAI); 2022. doi: 10.1109/seai55746.2022.9832076
46. Song W, Zhang G. Creative Thinking Stimulates Innovation Design of CAM Mechanism. In: Proceedings of the 2023 16th International Symposium on Computational Intelligence and Design (ISCID); 2023. doi: 10.1109/iscid59865.2023.00020
47. Tarek A, Alveed A, Farhan A. A Proof Theoretic Exploration of Mathematical Induction in Computational Paradigm. In: Proceedings of the 2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE); 2023. doi: 10.1109/csce60160.2023.00162
48. Mei Y, Ge Y, Zhang Y, et al. Research on Drilling and Completion Design Knowledge Base System based on Knowledge Map. In: Proceedings of the 2022 IEEE 2nd International Conference on Electronic Technology, Communication and Information (ICETCI); 2022. doi: 10.1109/icetci55101.2022.9832121

49. Pengna P, Leelasantitham A, Sukamongkol Y. A Low-Code Platform of Carbon Credit Trading Using Blockchain Technology: A Case Study in Nakhon Si Thammarat Province. In: Proceedings of the 2024 5th Technology Innovation Management and Engineering Science International Conference (TIMES-iCON); 2024. doi: 10.1109/times-icon61890.2024.10630773
50. Dolatabadi SH, Gatial E, Budinská I, et al. Integrating Human-Computer Interaction Principles in User-Centered Dashboard Design: Insights from Maintenance Management. In: Proceedings of the 2024 IEEE 28th International Conference on Intelligent Engineering Systems (INES); 2024. doi: 10.1109/ines63318.2024.10629098
51. Asakawa K, Tanaka T. Visual Programming Environment for Learning Functional Programming Using Unit Test. In: Proceedings of the 2022 12th International Congress on Advanced Applied Informatics (IIAI-AAI); 2022. doi: 10.1109/iiaiaai55812.2022.00051
52. Hourani H, Wasmi H, Alrawashdeh T. A Code Complexity Model of Object Oriented Programming (OOP). In: Proceedings of the 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT); 2019. doi: 10.1109/jeeit.2019.8717448
53. Kiczales G, Theimer M, Welch B. A new model of abstraction for operating system design. In: Proceedings of the Second International Workshop on Object Orientation in Operating Systems; 1992.
54. Marron M. A new generation of intelligent development environments. In: Proceedings of 2024 IEEE/ACM First IDE Workshop (IDE); 2024.
55. Xian C, Fu M. Towards a Taxonomy of Human-Computer Interaction (HCI) Methods Based on a Survey of Recent HCI Researches. In: Proceedings of the 2022 IEEE 2nd International Conference on Power, Electronics and Computer Applications (ICPECA); 2022. doi: 10.1109/icpeca53709.2022.9718950
56. Piva FJ, de Geus D, Dubbelman G. Empirical generalization study: Unsupervised domain adaptation vs. domain generalization methods for semantic segmentation in the wild. In: Proceedings of 2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV); 2023.
57. Yang C, Huang R, Yu X, et al. Math Word Problem Solver Based on Text-to-Text Transformer Model. In: Proceedings of the 2021 IEEE International Conference on Engineering, Technology & Education (TALE); 2021. doi: 10.1109/tale52509.2021.9678686
58. Meng H, Yang T, Yu X. A Bi-Channel Math Word Problem Solver With Understanding and Reasoning. In: Proceedings of the 2021 IEEE International Conference on Engineering, Technology & Education (TALE); 2021. doi: 10.1109/tale52509.2021.9678542
59. Gandhi J, Gandhi P, Gosar A, et al. Natural Language Processing based Math Word Problem Solver and Synoptic Generator. In: Proceedings of the 2022 3rd International Conference on Electronics and Sustainable Communication Systems (ICESC); 2022. doi: 10.1109/icesc54411.2022.9885451
60. Mandal S, Naskar SK. Classifying and Solving Arithmetic Math Word Problems—An Intelligent Math Solver. IEEE Transactions on Learning Technologies. 2021; 14(1): 28-41. doi: 10.1109/tlt.2021.3057805
61. Sepulveda F, Fan Y, Gabbianelli C, et al. Using ACT-R Architecture in the Design of Intelligent Tutoring Systems for VR Training of Manufacturing Skills. In: Proceedings of the 2024 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct); 2024. doi: 10.1109/ismar-adjunct64951.2024.00028
62. Kim B, Lee J. Analysis of Enumeration Strategy Use in the ACT-R Cognitive Architecture. In: Proceedings of the Third International Conference on Natural Computation (ICNC 2007); 2007. doi: 10.1109/icnc.2007.235
63. Khandan H, Lucas C. Implementing an XML based Unified Knowledge Manipulation Languages for SOAR cognitive architecture. In: Proceedings of the 2008 IEEE Conference on Cybernetics and Intelligent Systems; 2008. doi: 10.1109/iccis.2008.4670859
64. Zhu P, Lv B, Zou W, et al. Construction of super large interpretable machine intelligence system. Computer technology and development. 2024; 34(11): 172-179. doi: 10.20165/j.cnki.ISSN1673-629X.2024.0232