Personal Autonomous Intelligence Computer

Hanqing Wu [1] **Zile Yang** [2] **Jiahuan Zhang** [2] **Dongbai Chen** [1] **Baoping Hao** [1] **Hao Zhang** [1] **Hongying Han** [1] **Wenlin Fu** [1] **Kaicheng Yu** [1 2]

Abstract

In this position paper, we introduce a concept of personal autonomous intelligence computer (PAIC), to combine the advantage of two mainstream intelligence systems, connectionism and rule-based symbolism. Similar to a traditional von Neumann architecture, our PAIC consists of the input-output, memory, control, processing, and storage Unit. PAIC not only removes the limitation of the fix-length context window from the large language model but also achieves multi-hop reasoning-related precise manipulation of input with a novel pipeline. We also argue that, with the progressive increase of PAIC users on the Internet, the data flow of the current Internet will be reformed to achieve a decentralized local network. Our project page is publicly accessible at `https://github.com/KMind-Inc/PAIC`

1. Introduction

Recent mainstream AI systems primarily utilize Large Language Models (LLMs), exemplars of connectionism, for their notable generalization capability (Xi et al., 2023). This approach facilitates general intelligence in diverse fields such as text-to-image generation (Kim et al., 2023; Ge et al., 2023), tool-use (Qin et al., 2023; Patil et al., 2023), research (Bran et al., 2023; Boiko et al., 2023), etc.

However, the connectionism paradigm fundamentally involves a probabilistic model, inherently susceptible to errors, notably the "hallucination" phenomenon (Xu et al., 2023). In other words, due to the intrinsic structural design of LLMs and their reliance on neural network architectures, achieving precise data control solely through LLMs is inherently insufficient. If we provide paragraph excerpts from the classical English literature, *Harry Potter and the Philosopher's Stone* (Rowling, 2015), to the ChatGPT-4 (OpenAI, 2024), which is the state-of-the-art LLM-based AI system, and repeatedly ask it the following question, *"How many words or phrases which indicate the Muggle characters are directly and indirectly mentioned in the provided passage? Please return a number only."*, [1] it consistently failed to provide the correct answer.

The complexity of this query stems from the necessity for an AI system to first identify and comprehend the context related to the "Muggle character" within the provided passage, in order to understand what the "Muggle character" is. Subsequently, it is required to interpret the entirety of the text to deduce which words or phrases align with the definition of the "Muggle character". Finally, following the user specifications, "Please return a number only.", it is tasked with accurately enumerating the number of words or phrases implying the "Muggle character", and subsequently relaying the cumulative count back to the user. This indicates that the state-of-the-art LLM still struggles to precisely understand and manipulate the fine-grained structure of data, and we categorize it as *AI Precision* problem. To achieve the capability of data manipulation with finer granularity, we advocate for an AI system capable of multi-hop reasoning. For this "Muggle" Counting problem which is multi-hop reasoning-based, our system correctly identify all words or phrases which directly or indirectly indicate the meaning of the "Muggle character" [2]

Conversely, the issue of hallucination is fundamentally nonexistent in the context of symbolic approaches. This is because the reasoning capability of the symbolism is based on explicitly set rules and logical operations (Zhang et al., 2021). In other words, symbolic methods utilize powerful declarative languages for knowledge representation, offering clear, interpretable reasoning processes (Zhang et al., 2021). Nevertheless, the symbolism paradigm's reliance

[1]KMind Technology Co., Ltd., Hangzhou, China [2]Autonomous Intelligence Lab, School of Engineering, Westlake University, Hangzhou, China. Correspondence to: Kaicheng Yu <kyu@kmind.com>.

[1]Please refer to the Figure 7 from Appendix B for the complete example. Note that an example including all the deductive clues, one phrase ("Alice Qu") directly and two words ("who", "She") indirectly mentioning the Muggle character ("Alice Qu"), has also been provided when prompting such systems, and the data-analysis plug-in that explicitly designed to handle user-uploaded texts has also been enabled.

[2]Please refer to the Figure 8 in Appendix B for a detailed explanation.

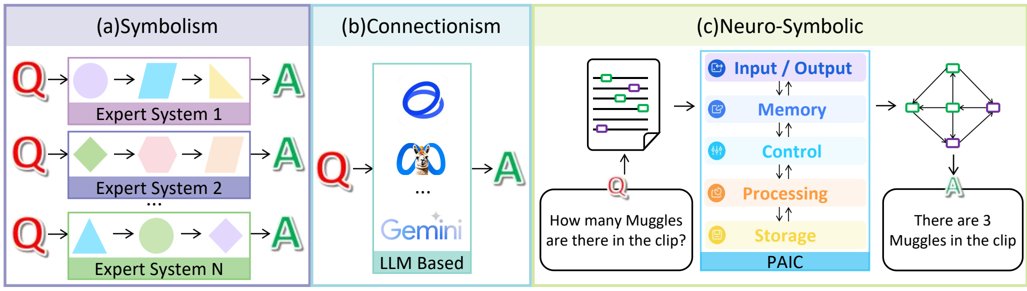

Figure 1. **Paradigm of symbolism, connectionism and our neuro-symbolic solution.** For the resolution of the multi-hop reasoning-related queries, solutions can generally be categorized into three distinct paradigms, symbolism, connectionism, and neuro-symbolic: (a) representatives of symbolism, expert system (Wang et al., 2022a), primarily relies on the construction of a series of hand-crafted rules to facilitate reasoning in a specific case, hereby, lacking in generalizability; (b) quintessential embodiment of connectionism, Large Language Model (LLM), exhibits robust generalization capability in general inferential problems but always suffers from the hallucination problem (Xu et al., 2023), failing to generate the precise answer; (c) our solution, *Personal Autonomous Intelligence Computer*, composed of a set of distinct units, takes full advantage of both the symbolism and the connectionism, large language model, forming the structure data which contains the high-dimensional information, such as the knowledge graph to first translate an arbitrary user input into a verifiable graph structure so to resolve uncertainty introduced by generative connectionism pipeline.

on a top-down, rule-based strategy limits its generalizability, necessitating substantial manual adjustments to tailor rules for diverse input scenarios. Hence, to attain a higher precision in data manipulation capability, we propose our solution, Personal Autonomous Intelligence Computer (PAIC), integrating the strengths of connectionism's generalization capabilities with the precise data control offered by symbolism.

As depicted in the middle block of the Figure 1(c), our PAIC consists of common components in the Von Neumann Architecture (Von Neumann, 1993). Compared to the early approaches that only add an external memory unit to extend the context window (Packer et al., 2023), our PAIC aims to leverage a graph-driven approach to alleviate the hallucination. Specifically, we treat the large language model like a central processing unit of a traditional computer and decompose the reasoning tasks into various sub-tasks to avoid excessively using highly complex prompts. We then propose a novel data pipeline, dehydration-verification-rehydration, to realize a close-loop verification process to ensure the correctness of our PAIC's execution.

Besides, our PAIC is also characterized by customized storage and user-defined capabilities. Specifically, deploying our product in a cloud computing environment not only liberates it from the fixed-length context window limitation of LLMs but also provides expandable storage capacity. Furthermore, unlike the domain-specific language models, we aim to build an operating system, dubbed kOS, that contains some common operations like parsing online content of web pages and also supports user-defined ones to ease the customization. To this end, we can easily reorganize these operations into lightweight downstream applications.

As user numbers grow and considering our PAIC's continuous online presence, we hypothesize that an evolving network framework, building incrementally on current Internet architecture, will gradually emerge. We dub this novel network paradigm the *AI Internet*. More significantly, the advent of the AI Internet implies that the paradigm of information dissemination may undergo alterations. Current Internet architecture funnels a significant amount of data to platforms providing information alignment services, as evidenced by the growing preference for online shopping companies like Amazon and Alibaba. These information-matching service-offering platforms align a boarder range of options with the user requirements, including some unique information such as the off-price merchandise. Several third-party platforms mandate their partner merchants to provide the lowest prices exclusively on their platforms, often resulting in artificially inflated prices on competing platforms (Com, 2021; b, 2021; Sheng et al., 2021). In essence, these platform companies monopolize a market segment by controlling both demand and supply information. In contrast, people can transmit their requests across the AI Internet through their PAICs. Subsequently, other PAICs possess the eligible answer will automatically reply to those PAICs that broadcast requests. Hence, the data eventually flows towards the demands instead of the third-party platform. In summary, our vision is:

- We propose an initial design structure of Personal Autonomous Intelligence Computer (PAIC), taking full advantage of the symbolism and connectionism, which are characterised by supporting the precise operation with multimodal data and the user-defined capability to alleviate the current hallucination problem of current generative models.

- We hypothesize a new paradigm of the internet will emerge with the growing number of our PAIC (or equivalently other types of agent-based systems) users, that the information will not inevitably flow to tech-giants but decentralized local hosts. As such, it sheds light on resolving the current data monopolization phenomenon.

2. Personal Autonomous Intelligence Computer (PAIC)

As mentioned above, with the aim of achieving precise data manipulation, our solution is Personal Autonomous Intelligence Computer (PAIC). This section will illustrate how to accomplish this functionality built on the architecture of our PAIC.

2.1. Architecture

Figure 2 illustrates that the PAIC architecture preserves the core elements of the traditional Von Neumann architecture (Von Neumann, 1993) rooted in the Turing Machine Model (Turing et al., 1936). Specifically, PAIC is composed of:

Input-Output (IO) Unit. As the purpose of our PAIC is to accurately understand and manipulate personal data, we shall support various input sources, such as traditional application (APP), Applet services, instant messager (IM) and its modalities include text, audio, images and videos. We also support a language user interface (LUI) which leverages a customized langauge model to process arbitrary user commands.

Control Unit. In our design, this unit serves as the 'brain' of the entire PAIC system. It comprises a *Decision Maker* for task and strategy selection, an *Evaluator* for performance assessment, and a *Job Manager* for task allocation, collectively executing precise control in accordance with user demands. Its internal design facilitates logical recursion, mirroring the human problem-solving process.

Processing Unit. It primarily consists of the *Language Module* and the *Job Executor*. The Job Executor is mainly responsible for the task execution, which can also leverage the external tools library to support arbitrary user-defined operations. Note that, the *Language Module* can be called by other units, for example, the IO Unit will use it to understand and decompose the user's language instruction input.

Tools Library. This unit mainly consists of external tools with two features, an automatic tool learning approach with Robotic Process Automation (Madakam et al., 2019) to support cross-platform applications, and a zoo of predefined tools pipeline, e.g. Photoshop and After Effects for photo and video editing.

Memory Unit. Similar to a traditional computer, the memory unit is designed to store the content of an active session when processing the user's input command, including the embedding of various input formats, like text, images, videos and audio.

Storage Unit. Unlike the memory unit, this unit is permanent data storage, such as previous conversations, and external uploaded data, so to let PAIC understand the user's patterns and preferences. Note that, the content of memory and storage units can be updated accordingly with the help of our processing unit.

2.2. kOS: An operating system for PAIC

An operating system (OS) is crucial for a computer, as it bridges computer and software architectures, manages core system functions, and facilitates user interaction and application execution. (Wilkes & Needham, 1979; Unwana et al., 2022). Similarly, we introduce, kOS, an operating system developed on our PAIC architecture. By analogy with the software and applications executed within the current OSs, a novel concept, *Action* (ACT). In our kOS, ACT is a certain program to combines certain machine learning operations with predefined logic to achieve certain functions. We categorize such ACT into three kinds of levels: native ACT, system ACT and customized ACT. *Native ACT (NACT)*, standing for the atomic functionalities of our PAIC that are launched on kOS, NACT can thus be considered kOS's minimal execution unit, encapsulating the interaction paradigm among PAIC components in line with their essential functionalities. As illustrated by the Figure 3, it is evident that both the *System ACT (SACT)* and the *Customized ACT (CACT)* are implemented through a series of logical combinations based on pre-encapsulated NACTs, facilitating more complex capabilities.

Native ACT (NACT). NACT represents the smallest executable unit within our kOS. It encapsulates collaborative interactions among PAIC components, aligning with their core functionalities. For example, according to the Figure 3, the invocation of NACT *k_get_act_request()* mainly relies on the collaboration between the IO and Processing Units, designed to decipher user intentions through query deconstruction.

System ACT (SACT). SACT is executed using a series of logical operations grounded in pre-encapsulated NACTs, facilitating enhanced capabilities. Taking the SACT *weblinkSummarise()* as an example, it essentially hinges on the NACT *k_web_get_page_data()* for obtaining the content of the input web link, *k_semantic_summarize()* for synthesizing the core concept of the input text, thereby, mainly in charge of summarizing the core content from the input web link. We illustrate the utilization example and code implementation of this SACT in Figure 17 and Figure 18.

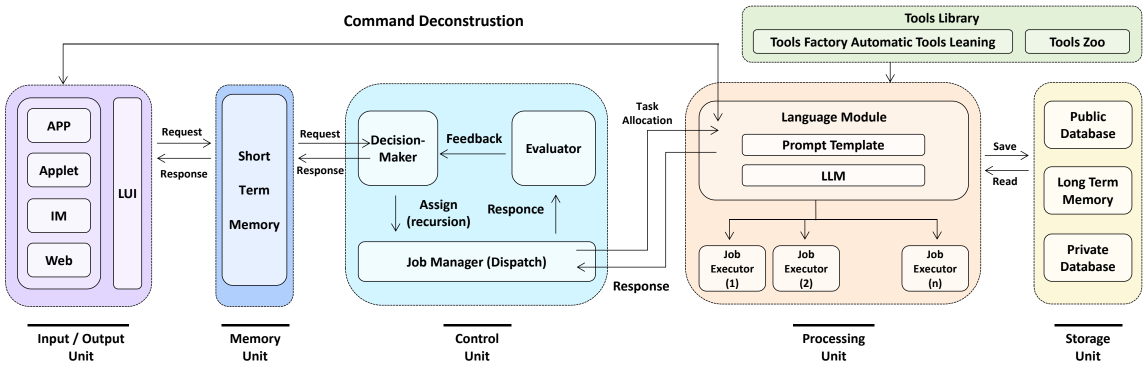

Figure 2. **Infrastructure Design of Personal Autonomous Intelligence Computer (PAIC)**

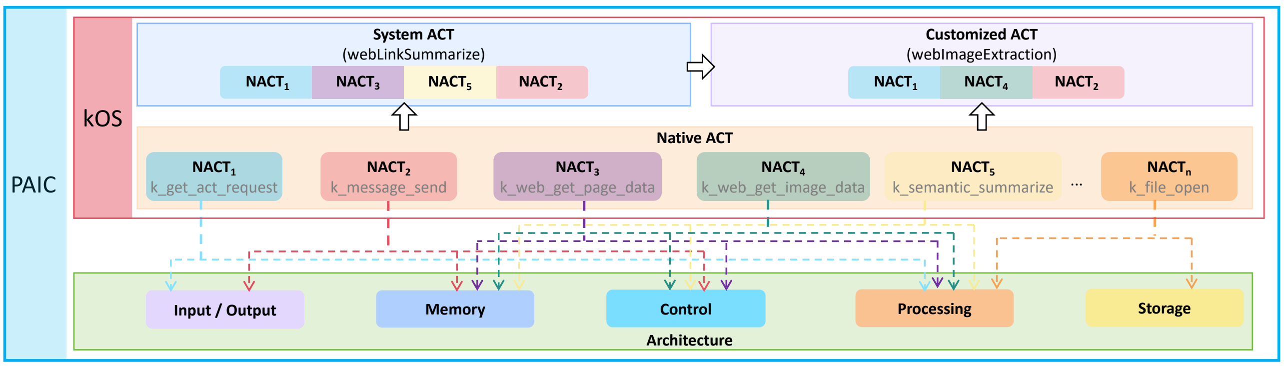

Figure 3. **Overview of PAIC and its operating system (kOS)**. Here, we delineate the relationship between PAIC and kOS. Similar to a traditional operating system, our kOS runs on top of PAIC to connect users' requests with low-level functions. There are two layers of kOS, Natice ACT that serves as basic operations and directly uses infrastructure units, and System ACT & Customized ACT which are composed of native ACT.

Customized ACT (CACT). In essence, CACT can be regarded as an extension of the SACT, typically implemented by invoking the corresponding NACTs and SACTs if necessary through a series of logical rules, thereby enabling more intelligent functions to meet personal requirements. Taking the CACT *webImageExtraction()* from Figure 3 as an example, this CACT is to extract the images from the input weblink, essentially relying on the NACT *k_web_get_Image_data()*

Please refer to Appendix C for a detailed explanation of all NACTs and SACTs.

3. Data Manipulation

In order to delineate the process of the ultimate target of our PAIC, data manipulation with finer granularity, more comprehensively, this section will primarily dissect an example. In our system, operable data objects are categorized into four types: text, images, video, and audio. Note that, we understand that processing arbitrary user input is nearly impossible at this current state, so we set a limited context, defined by our System ACT or Customized ACT, for the following manipulation procedure. We believe that processing arbitrary language inputs given such limitations as desired is possible.

In Figure 4, it is distinctly evident that the entire process of the data manipulation is segmented into three phases [3] , **Data Dehydration (DD)**, **Data Structure Equilibrium Recovery Recursion (DSERR)**, and **Metadata Rehydration (MR)**. These three phases connected sequentially, **DD** $\rightarrow$ **DSERR** $\rightarrow$ **MR**, defines a novel paradigm to accomplish data manipulation with finer granularity. In response to this novel paradigm, we propose another novel comprehensive concept for measuring the overall performance, *AI Precision*. AI Precision is fundamentally a composite evaluation metric, which can be decomposed into the evaluation metrics for both the DD and DSERR phase performance

[3] All these novel concepts are derived from the Three-Body Problem (Liu & Liu, 1976), which are proposed to aid users in better comprehending the essence of each phase.

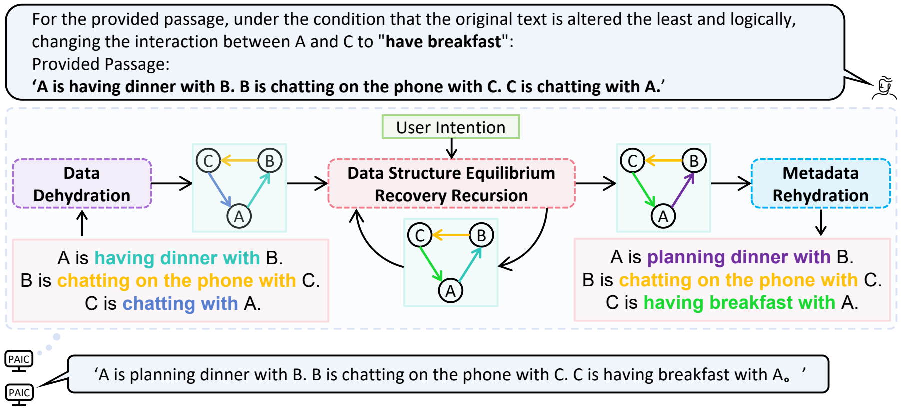

Figure 4. **A novel paradigm for the precise data manipulation.** A passage is converted to high-dimensional structure data first in the **Data Dehydration** phase. The target edge and the target interaction obtained from the raw user query are input to the following phase, forming the new structured data that destroyed its original equilibrium status. The new structured data will be recursively updated until it returns to the equilibrium status. For the last phase, the input structured data in equilibrium will be converted back to the text.

measurement.

Additionally, all the concepts and notation utilized for the later explanation are defined below:

Node. Within the scope of this paper, it can be divided into the subject node and the object node. For example, "A is having dinner with B.", the subject node represents "A"; the object node is "B".

Target Edge. It is the edge that is directly relative to the user query. Note that, some queries may contain more than one target edge. Specifically, we use $e^{i \to j}_{\text{Target}}$ to represent the Target Edge from node i to node j.

Relevant Edge. It refers to all the edges connected to the subject nodes and the object nodes of the "Target Edge", excluding the "Target Edge" itself. Similar to the definition of the Target Edge, we use $e^{i \to j}_{\text{Relevant}}$ to represent the Relevant Edge from node i to node j.

Irrelevant edge. Conversely, all the other edges not defined by the scope of the "Target Edge" and the "Relevant Edge" are classified as the "Irrelevant Edge".

Interaction. For example, "A is having dinner with B.", the interaction within the scope of this paper is "have dinner".

3.1. Data Dehydration

Referring to the example shown from the Figure 4[4], the provided passage from the user query is "A is having dinner with B. B is chatting on the phone with C. C is chatting with A". The key step of this phase is to generate the high-dimensional structured data, which is a graph structure composed of the node and edge in our example. All nodes contain a subject and its coordinates in the format of "(sentence order, subject order in this sentence)". All edges encapsulate information, excluding node details, like interactions between subjects and objects, auxiliary verbs, main verbs, and prepositions. Note that, one node can be the subject of one interaction or/ and the object of another interaction. The node coordinate is stored only when the node serves as the subject. All the edges are directional, pointing from the subject to the object.

Additionally, if provided, the corresponding attribute information for both the node and the edge will also be excavated. For example, any information pertaining to the node, such as basic background, relationships, categories, etc., will be documented. Similarly, the corresponding information relative to the original text, such as the sentence tense, the main verb, the auxiliary verb, the preposition, etc., will be recorded as the edge information. Unlike many knowledge graphs tailored for specific uses like relation extraction (Weston et al., 2013; Riedel et al., 2013), semantic parsing (Berant et al., 2013; Heck et al., 2013), and question-answering (Bordes et al., 2014b;a), etc., our graph structure supports not only a variety of tasks related to explicit or implicit relational reasoning but also the reverse operation of complete original text recovery.

In more complex scenarios involving image, video, or audio data, the storage of high-dimensional structured data often relies on the method employed for its generation, and may require formats such as databases. Hence, a distinctly evident bottleneck is the capability of the high-dimensional structured data generation. The greater the capability to generate high-dimensional structured data, the finer the granularity of the data available for subsequent operations. In general, the approaches from the data mining area would be more suitable for this sort of problem (Shu & Ye, 2023).

[4]You may also see Figure 6 in Appendix A for the complete process and detailed explanation.

Furthermore, proposing a suitable evaluation metric, such as the information entropy-like metric, for measuring how much information both the different and the same high-dimensional structured data can contain is essential to find out the optimal high-dimensional structured data. We leave all these potential improvements in our future work.

3.2. Data Structure Equilibrium Recovery Recursion

The design of this phase is inspired by the reflection steps undertaken by humans in the problem-solving process. The most important step in this phase is to detect whether the structured data is in equilibrium, which is undertaken by the evaluator. More importantly, the criteria for achieving equilibrium are defined based on the user's query.

Following the example depicted by the Figure 4, the condition for achieving equilibrium is the absence of logical conflicts in both temporal and spatial dimensions. [5]

In our example, "Target Edge" set contains the $e^{C\rightarrow A}_{\text{Target}}$ only; "Relevant Edge" contains the $e^{A\rightarrow B}_{\text{Relevant}}$ and the $e^{B\rightarrow C}_{\text{Relevant}}$; there aren't any "Irrelevant Edge". Furthermore, the target edge and interaction identified in the preliminary step also serve as inputs for the current phase. Hence, by replacing the original interaction of the target edge with the target interaction ("chatting with"→"having breakfast with"), the equilibrium of the original structured data is disrupted. The following step is to evaluate whether the current structured data is in equilibrium. More specifically, all "Relevant Edges" are evaluated by the Language Module from the Processing Unit for potential temporal and spatial logical conflicts with the "Target Edge". All "Relevant Edges" exhibiting this type of logical conflict, will be fed into the Language Module for updating. Each of these "Relevant Edges" will be allocated a set of reasonable interactions.

In order to achieve the capability of data manipulation with finer granularity, we also introduce a concept to quantify the degree of disruption to the original text, counting the number of words changed from the original text. Hence, all the corresponding "Relevant Edge" will be eventually updated to an optimal "Edge" with the minimum number of words changed from the original text. All these updated "Edges" will then become the new "Target Edge" for the next recursion. Similarly, they follow all the aforementioned steps to update their own "Relevant Edges" exhibiting the same logical conflict. Note that, all the edges that had been classified as the "Relevant Edge" will not be considered as the "Relevant Edge" in the following process. The recursion terminates only when no more "Relevant Edges" are found that have temporal and spatial dimensional logical conflicts with the current "Target Edge". At this point, the resultant structured data can be regarded as returning to the equilibrium status, which will be output to the final phase.

Based on the aforementioned description, it is not difficult to observe that the equilibrium condition plays a pivotal role in this phase. For this example, it mainly considers the logical conflict in the temporal and spatial dimensionality. Currently, all the conditions are composed of handcrafted rules according to the different applications. We emphasize that this equilibrium condition is defined by the user's defined context. In the future, we propose to self-adaptively generate the equilibrium conditions via the user query, enhancing the generalization capability of the evaluator.

3.3. Metadata Rehydration

This is the last phase of the data manipulation process, which mainly focues on recovering the text from the structured data in this example. As characteristics of such structure data, the coordinates of all the subject nodes had been stored in the DD phase. In addition, the rule for recovering the text from the structured data has also been predefined according to the user-provided passage. Therefore, all the raw sentences from the original text can be easily retrieved via our PAIC processing unit.

Our final answer is "A is planning dinner with B. B is chatting on the phone with C. C is having breakfast with A", where the cost is 2 words changed. Given our example, the answer from ChatGPT-4 as of January 8th, 2024, was completely erroneous and lacked logical coherence, which is shown by the Figure 13 in Appendix B. Additional testing examples have also been provided for the same version of the ChatGPT-4. None of the answers provided by ChatGPT-4 is reasonable, which can be viewed from the Figure 14 and Figure 15 in Appendix B as well. Later on, another more challenging example related to four nodes is again fed to the ChatGPT-4 and our PAIC. Referring to the Figure 11, it is evident that ChatGPT-4 still fails to generate a reasonable answer[6]. Conversely, PAIC's answer seems more reasonable, which can be viewed from the Figure 12 in Appendix B.

[5] For example, providing two sentences, "A is having dinner." and "A is having breakfast.", the subject of both sentences is "A". The tense of both sentences is the simple present tense. The event of the former sentence is "having dinner", while, that of the later one is "having breakfast". Therefore, we define existing the logical conflict between these two sentences in both temporal and spatial dimensions because we all know it is impossible for the same person to have dinner and breakfast at the same time.

[6] ChatGPT-4's answer: "A is having dinner with B. B is chatting on the phone with C. C is chatting with A. D is having breakfast with A. D is having dinner with C." Obviously, "D" is impossible to have dinner and breakfast at the same time, which is irrational. Additionally, please refer to Appendix B for a detailed comparison of the multi-hop reasoning-based information replacement task between ChatGPT-4 and our PAIC

3.4. AI Precision

We hypothesize that the composite concept, AI Precision, for textual data, should combine metrics for the information in structured data with those quantifying textual disruption and provide an initial version. However, we believe there exists much room for future improvement in this direction.

For the DD phase, the quality of the high-dimensional structure data directly determines the granularity of subsequent operable data. The greater the amount of information encapsulated in this high-dimensional structured data, the finer the granularity that can be achieved in data manipulation. Hence, a metric capable of measuring the information encapsulated in this structured data can expedite the process of identifying the optimal structured data configuration.

For the DSERR phase, in our example, we quantify textual disruption by the most intuitive metric: the number of words altered from the original text. We posit that undiscovered metrics, considering factors like story logic rationality, narrative arc integrity, inter-sentential correlation, and sentence coherence, could enhance tasks involving insertion, deletion, modification, and retrieval of textual data. Integrating these undiscovered metrics enables a more comprehensive quantification of textual disruption, vital for optimizing system responses.

4. AI Internet: Connecting All The PAICs

We hypothesize that with the increasing number of PAIC users in the current Internet space, the data flow will undergo a subtle change. We start by discussing the issues of data monopolization and privacy in modern Internet society, and outline how personal AI devices, such as our PAIC, can help with these concerns. We hope to shed light on a future where the additional value of our personal data is not exploited by tech giants, but instead returned to ourselves.

4.1. Data Monopolization and Data Privacy

Monopolization has become inevitable in the current structure of Internet society. As analyzing the entire Internet involves unnecessary complexity, we construct a simplified 'Internet' in the left part of Figure 5. Each node either be a shop or a customer. Imagine that one customer has a simple request: 'buy a diaper at the lowest price'. To this end, you need to inquire with all shops about the availability of diapers and then request pricing information. As ordinary individuals, we recognize that our time for this task is limited. Therefore, the most efficient method involves creating a super-node that aggregates all information from each store-node, allowing the customer to interact solely with the super-node. In reality, these super-nodes are the tech giants possessing the computational capacity to process all local nodes, thereby deriving profit either directly from each query or indirectly through advertisement.

However, this paradigm might lead to several downsides for customers. Firstly, although a single data record may appear insignificant, the growing number of customers enables these super-nodes to amass vast amounts of user-node interactions, including browsing histories, purchase records, and more. Furthermore, without resorting to selling this data to third parties, these super-nodes can offer targeted advertising services to commercial nodes such as the shop-node, thereby generating significant profits. In reality, these services constitute one of the most lucrative strategies employed by these giants. This phenomenon, characterized by data flowing from the end-user to the super-node while end-users seek useful information, is termed 'user-seek-data' principle.

4.2. The Reversal of the Information Paradigm

While the societal impact of data harvested by super-nodes remains contentious, we explore an alternate outcome arising with the integration of personal computational devices, like our Personal Autonomous Intelligence Computers (PAIC), into the Internet. Following the aforementioned case, we argue that there might be two reasons why super-nodes will inevitably emerge: i) in a graphical network, the cost of broadcasting a request to the entire graph quadratically increases with the number of nodes; ii) each end-node has limited computational resources to process such large-scale data. In a traditional internet framework, each customer node typically represents an individual using a device such as a web browser, where manually comparing prices and quality is evidently tedious. We conjecture that this rationale underpins data monopolization. However, the advent of generative large language models is likely to shift this paradigm.

Envision a scenario where end-users, equipped with our personal autonomous intelligence computer (PAIC) or similar devices capable of accurately understanding user requests, could bypass the traditional sequential manual process. First, with the parallel support of our PAIC, users can effortlessly broadcast their requests to each shop node, enabling swift execution and summarization of price comparisons. In addition, our PAIC offers a language user interface, allowing end-user to customize their request. In this scenario, by merely sending anonymous links, such as using a web browser's privacy mode to conceal digital footprints, users can significantly ensure their data privacy. Consequently, the optimal strategy for the super-node involves reporting accurate prices. In this paradigm, the autonomous execution of broadcasting and comparison allows data to flow directly to users with minimal human intervention. Hence we dub this 'data-seek-user' and this new paradigm *AI Internet*.

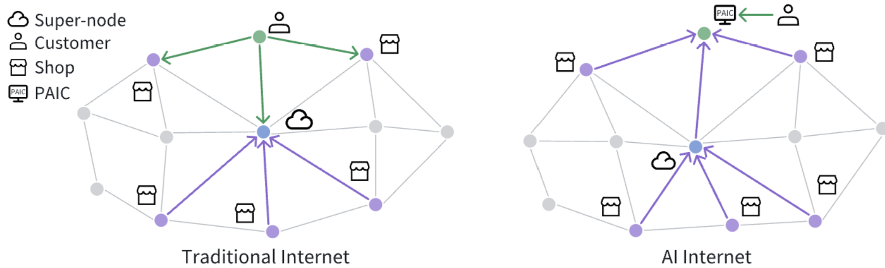

Figure 5. **(Left) Traditional Internet.** We formulate a small 'Internet' with three types of nodes, customer, shop and super-node. Assume a customer has a purchase request, they need to manually go through (denoted with green arrow) either the shop with known addresses or visit super-node who processes (denoted with purple arrow) the categories of many shops. As one person usually has limited time, the most efficient approach is to visit the super-node as much as possible which inevitably leads to data monopolization. **(Right) AI Internet.** However, if the customer is equipped with a personal computing device with the autonomous ability to browse the web, such as our PAIC, they would only need to communicate with the PAIC while PAIC takes all the search and communication to the shop-nodes given sufficient computational resources. In general, this leads to a paradigm that information flows towards a person with PAIC to largely improve the efficiency of seeking information.

5. Literature Review

In this section we will introduce background knowledge closely related to the implementation of PAIC.

5.1. Von Neumann Architecture

Until now, the development of digital computers has been grounded in the Turing machine model (Turing, 1936). Turing introduced the stored program concept, wherein a Turing machine's description number serves as both storable data and an executable program for a universal Turing machine. Subsequently, John von Neumann was inspired by Turing's ideas and designed a structure that separated storage and calculation in the EDVAC project (Von Neumann, 1945), called the von Neumann architecture. It established the foundational architecture of today's manufacturable computers. Today's computers fundamentally embody the von Neumann architecture. Similarly, our PAIC, which retains all components of the von Neumann architecture, is evidently a computer.

5.2. Operating System

The operating system, serving as the human-computer interface, manages the allocation and scheduling of the computer's software and hardware resources (Bullynck, 2018; Tanenbaum & Woodhull, 1997). It evolved from early systems based on punch cards and tapes, through the development of mainframes and microcomputers, and gradually transformed into modern graphical user interface operating systems such as Windows, Mac OS, and Linux (Field, 2022; MEI et al., 2022; Chakraborty, 2023). This entails a continuous quest for an enhanced, user-friendly experience. Different from traditional operating systems, our kOS controls PAIC by triggering NACTs via human language, enabling more natural computer interaction.

5.3. Large Language Model Based Reasoning

The growing intelligence in LLMs has increasingly focused research on their use for complex reasoning tasks. A key focus is on utilizing advanced prompting methods to reduce hallucination issues in LLMs (Wei et al., 2022; Wang et al., 2022b; Liu et al., 2021; Petroni et al., 2019), etc. Another prevalent approach involves fine-tuning LLMs with domain-specific data featuring similar characteristics (Hu et al., 2023). Essentially, these works significantly increase the probability of aligning LLM behaviour with human preferences. Nevertheless, the phenomenon of hallucination has yet to be eradicated. Hence, our PAIC employs the core idea from the neuro-symbolic system to mitigate this unavoidable issue.

5.4. Knowledge Graph

Knowledge graphs have shown efficacy in enhancing user queries with relevant concepts, such as through Entity Query Feature Expansion (EQFE), which enriches query information using the query knowledge graph (Dalton et al., 2014). Consequently, knowledge graphs have been demonstrated to possess enhanced efficiency in search and retrieval tasks, thereby yielding more accurate answers in information retrieval-related endeavours, which is fundamentally one of the representations of our proposed concept, high-dimensional structured data.

6. Conclusion and Future Directions

In conclusion, we propose PAIC, defining a novel paradigm, DD phase$\rightarrow$ DSERR phase $\rightarrow$ MR phase, which supports the capability of multimodal data manipulation with finer granularity. AI Precision, for evaluating the overall performance of the task involving data manipulation based on this new-style paradigm, is also proposed. More importantly, each phase of this paradigm offers various research opportunities, mainly including identifying optimal high-dimensional structured data in the DD phase, adaptively establishing equilibrium conditions for any user query in the DSERR phase, and generating recovery rules in response to random user queries in the MR phase. We believe that these research topics hold instructive significance for the future design of more advanced AI systems.

Apart from that, we also propose an operating system built on our PAIC, kOS. We define the minimum execution unit of kOS, Native ACT, which encapsulates the interaction paradigm among PAIC components, aligning with their essential functionalities. Furthermore, kOS also supports the user-defined capabilities. Users can create their own ACTs, Customized ACTs, by invoking a series of Native ACTs to meet their specific requirements.

We hypothesize that the AI Internet will gradually evolve in tandem with the increasing number of PAIC users. In the future, AI Internet will be a decentralized network where information will no longer be tilted towards giants. Hopefully, this can provide a promising way to alleviate the issue of data monopolization.

References

Competitive effects of price parity agreements, Nov 2021. URL `https://www.compasslexecon.com/the-analysis/competitive-effects-of-price-parity-agreement/11-15-2021/`.

Price parity clauses and digital platforms: the rocky path to much needed clarity, March 2021. URL `https://www.linklaters.com/en/insights/blogs/linkingcompetition/2021/march/price-parity-clauses-and-digital-platforms-the-rocky-path-to-much-needed-clarity`.

World news for students of english, Jan 2024. URL `https://www.newsinlevels.com/products/2024-doomsday-clock-level-3/`.

Berant, J., Chou, A., Frostig, R., and Liang, P. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1533–1544, 2013.

Boiko, D. A., MacKnight, R., and Gomes, G. Emergent autonomous scientific research capabilities of large language models. *arXiv preprint arXiv:2304.05332*, 2023.

Bordes, A., Chopra, S., and Weston, J. Question answering with subgraph embeddings. *arXiv preprint arXiv:1406.3676*, 2014a.

Bordes, A., Weston, J., and Usunier, N. Open question answering with weakly supervised embedding models. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part I 14*, pp. 165–180. Springer, 2014b.

Bran, A. M., Cox, S., White, A. D., and Schwaller, P. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*, 2023.

Bullynck, M. What is an operating system? a historical investigation (1954–1964). *Reflections on programming systems: Historical and philosophical aspects*, pp. 49–79, 2018.

Chakraborty, P. *Operating Systems: Evolutionary Concepts and Modern Design Principles*. CRC Press, 2023.

Dalton, J., Dietz, L., and Allan, J. Entity query feature expansion using knowledge base links. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, pp. 365–374, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450322577. doi: 10.1145/2600428.2609628. URL `https://doi.org/10.1145/2600428.2609628`.

Field, C. Complete history of the operating system, December 2022. URL `https://history-computer.com/complete-history-of-the-operating-system/`. Accessed: 2024-01-10.

Ge, S., Park, T., Zhu, J.-Y., and Huang, J.-B. Expressive text-to-image generation with rich text. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7545–7556, October 2023.

Heck, L., Hakkani-Tür, D., and Tur, G. Leveraging knowledge graphs for web-scale unsupervised semantic parsing. In *Proceedings of INTERSPEECH*, 2013.

Hu, Z., Lan, Y., Wang, L., Xu, W., Lim, E.-P., Lee, R. K.-W., Bing, L., and Poria, S. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*, 2023.

Kim, Y., Lee, J., Kim, J.-H., Ha, J.-W., and Zhu, J.-Y. Dense text-to-image generation with attention modulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7701–7711, October 2023.

Liu, C. and Liu, K. *The three-body problem / Cixin Liu*. Tor Books, New York, 1976.

Liu, J., Liu, A., Lu, X., Welleck, S., West, P., Bras, R. L., Choi, Y., and Hajishirzi, H. Generated knowledge prompting for commonsense reasoning. *arXiv preprint arXiv:2110.08387*, 2021.

Madakam, S., Holmukhe, R. M., and Jaiswal, D. K. The future digital work force: robotic process automation (rpa). *JISTEM-Journal of Information Systems and Technology Management*, 16, 2019.

MEI, H., CAO, D., and XIE, T. Ubiquitous operating system: Toward the blue ocean of human-cyber-physical ternary ubiquitous computing mei. *Bulletin of Chinese Academy of Sciences (Chinese Version)*, 37(1):30–37, 2022.

OpenAI. Chatgpt (4). Large language model, 2024. URL `https://chat.openai.com`.

Packer, C., Fang, V., Patil, S. G., Lin, K., Wooders, S., and Gonzalez, J. E. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Patil, S. G., Zhang, T., Wang, X., and Gonzalez, J. E. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023.

Petroni, F., Rocktäschel, T., Lewis, P., Bakhtin, A., Wu, Y., Miller, A. H., and Riedel, S. Language models as knowledge bases? *arXiv preprint arXiv:1909.01066*, 2019.

Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

Riedel, S., Yao, L., McCallum, A., and Marlin, B. M. Relation extraction with matrix factorization and universal schemas. In *Proceedings of the 2013 conference of the North American chapter of the association for computational linguistics: human language technologies*, pp. 74–84, 2013.

Rowling, J. K. *Harry Potter and the philosopher's stone*, volume 1. Bloomsbury Publishing, 2015.

Sheng, W., Lee, E., Sheng, W., and Lee, E. What is 'forced exclusivity'? and why did it get alibaba fined $2.8 billion?, Jun 2021. URL `https://technode.com/2021/04/15/what-is-forced-exclusivity-and-why-did-it-get-alibaba-fined-2-8-billion/`.

Shu, X. and Ye, Y. Knowledge discovery: Methods from data mining and machine learning. *Social Science Research*, 110:102817, 2023.

Tanenbaum, A. S. and Woodhull, A. S. *Operating systems: design and implementation*, volume 68. Prentice Hall Englewood Cliffs, 1997.

Turing, A. Turing machine. *Proc London Math Soc*, 242: 230–265, 1936.

Turing, A. M. et al. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58 (345-363):5, 1936.

Unwana, T. E., Udoh, E. I., and Umoh, V. O. A study of the importance of operating system (os) in a computer system. 2022.

Von Neumann, J. First draft of a report on the edvac, 30 june 1945. *Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, PA, USA. Available online: https://library. si. edu/digital-library/book/firstdraftofrepo00vonn (accessed on 1 October 2022)*, 1945.

Von Neumann, J. First draft of a report on the edvac. *IEEE Annals of the History of Computing*, 15(4):27–75, 1993.

Wang, W., Yang, Y., and Wu, F. Towards data- and knowledge-driven artificial intelligence: A survey on neuro-symbolic computing. *arXiv preprint arXiv:2210.15889*, 2022a.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022b.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837, 2022.

Weston, J., Bordes, A., Yakhnenko, O., and Usunier, N. Connecting language and knowledge bases with embedding models for relation extraction. *arXiv preprint arXiv:1307.7973*, 2013.

Wilkes, M. V. and Needham, R. M. The cambridge cap computer and its operating system. 1979.

Wu, C. *Journey to the West*. People's Publishing House, 1980.

Xi, Z., Chen, W., Guo, X., He, W., Ding, Y., Hong, B., Zhang, M., Wang, J., Jin, S., Zhou, E., et al. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*, 2023.

Xu, W., Agrawal, S., Briakou, E., Martindale, M. J., and Carpuat, M. Understanding and detecting hallucinations in neural machine translation via model introspection. *Transactions of the Association for Computational Linguistics*, 11:546–564, 2023.

Zhang, J., Chen, B., Zhang, L., Ke, X., and Ding, H. Neural, symbolic and neural-symbolic reasoning on knowledge graphs. *AI Open*, 2:14–35, 2021.

A. Complete Data Manipulation Process

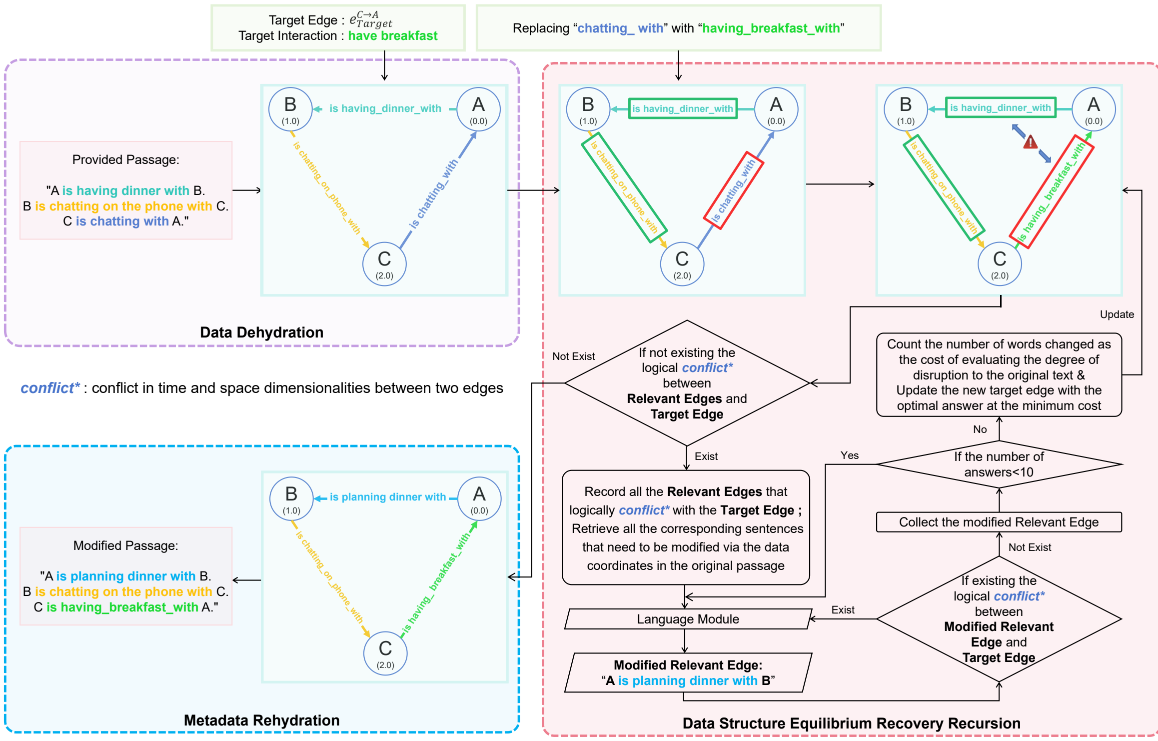

Figure 6. **Data Manipulation Process** for the information replacement task. The operable data object and the user intention were extracted from the preliminary step, thereby, obtaining the "Target Edge", $e^{C\to A}_{\text{Target}}$, "Target Interaction", "have breakfast", and the original text, "A is having dinner with B. B is chatting on the phone with C. C is chatting with A.". When all parameters are ready, the first step is to trigger the **Data Dehydration** phase, extracting the high-dimensional structured data which encompasses the complete information from the original text. The next step is to replace the original interaction "is chatting with" of the "Target Edge", $e^{C\to A}_{\text{Target}}$, with the "Target Interaction", "is having breakfast with", in order to meet the user requirement (The hidden step, not explicitly shown, involves using the Language Module to convert "have breakfast" into "is having breakfast with", retaining most of the original interaction's words.). With the establishment of the new "Target Edge", $e^{C\to A}_{\text{Target}}$, the balance of the original structure is disrupted, entering the **Data Structure Equilibrium Recovery Recursion** phase. All the "Relevant Edges" will then be stored first. As long as any one of them contains a conflict in the temporal and spatial domain with the "Target Edge", the Language Module will be employed to recursively generate a set of new "Interactions", where all of them can objectively coexist with that of the "Target Edge". In our system, we only generate ten reasonable answers for each "Interaction" needed to be modified. All updated "Relevant Edges" will then be updated to the new "Target Edge". In the next iteration, these new "Target Edges" will will search for any of their "Relevant Edges" that contain temporal and spatial logical conflicts with themselves. The aforementioned steps will be recursively performed until the stop condition is met, where no temporal or spatial logical conflicts exist between the current "Target Edge" and any of its "Relevant Edges". Upon satisfying the stop condition, the structured data is considered to have returned to equilibrium. Therefore, as the structured data reattains equilibrium, *Metadata Rehydration* phase is triggered. The high-dimensional structured data input will be reconverted into text.

B. ChatGPT-4's Responses Versus PAIC's Responses for the Multi-Hop Reasoning-Based Information Retrieval Task

This section mainly describes the ChatGPT-4 and our PAIC performance on the multi-hop reasoning-based information retrieval task. We provide two excerpts from the English literature, *Harry Potter and the Philosopher's Stone* (Rowling, 2015), and the Chinese literature, *Journey to The West* (Wu, 1980).

As illustrated by the Figure 7, it is obvious that the ChatGPT-4 equipped with the data analysis plugin only considers most of the word or phrase directly indicating the meaning of the 'Muggle character'. A majority of the pronouns such as *he, We, You*, are ignored. By contrast, as mentioned previously, our PAIC extracts the high-dimensional structure data from the provided text. This high-dimensional structure data is derived from the interconnection of all sentence-based node-edge graphs. Each node-edge graph encompasses the complete information of the corresponding sentence. Therefore, all words or phrases directly and indirectly conveying the "Muggle character"'s meaning can be easily retrieved via the classical graph traversal algorithm, such as breadth-first search, depth-first search, etc. Our PAIC correctly answers this "Muggle" Counting problem, which can be viewed from the Figure 8.

Similarly, when dealing with the "Wukong" Counting problem derived from an excerpt of *Journey to The West*(Wu, 1980), as depicted by the Figure 9, ChatGPT-4 successfully detects all the nickname ("XingZhe") of "Wukong", and the partial pronouns indirectly implying the meaning of "Wukong". In comparison with the answer of our PAIC which is shown by the Figure 10, the high-dimensional structured data is again extracted from the provided text. Thereby, the operable granularity of the text data has achieved a finer level, ending up with a correct answer to this "Wukong" Counting problem.

Prompt: How many words or phrases which indicate the Muggle characters are directly and indirectly mentioned in the provided passage? Please return a number only.
Provided Passage:
"Where was I?" said Hagrid, but at that moment, Vernon, still ashen-faced but looking very angry, moved into the firelight.
"He's not going," he said.
Hagrid, the teacher from the Magic School, grunted.
"I'd like to see how a Muggle like you stop him," Hagrid replied.
"A what?" said Harry, interested.
"A Muggle," said Hagrid. "It's what we call non-magic folk like them. And it's your bad luck you grew up in a family full of Muggles, except you and your parents"
"We swore when we took him in we'd put a stop to that rubbish," said Vernon, "swore we'd stamp it out of him! Wizard, indeed!"
"You knew?" said Harry. "You knew I'm a – a wizard?"
"Knew!" shrieked Aunt Petunia, in her husband Vernon's arms, suddenly. "Knew! Of course we knew! How could you not be, my dratted sister being what she was? Oh, she got a letter just like that and disappeared off to that – that school – and came home every holiday with her pockets full of frog-spawn, turning teacups into rats. I was the only one who saw her for what she was – a freak! But for my mother and father, oh no, it was Lily this and Lily that, they were proud of having a witch in the family!"

For example:
Provided Passage:
"Alice Qu is a lady who is in the white dress. She seems a bit weird."
Assume:
"Alice Qu" is a Muggle character inferred from the previous context.
Explanation:
For the provided text, the Muggle character "Alice Qu" is directly and indirectly
mentioned three times in total by the phrase ("Alice Qu"), and two words ("who", "She").
Final Output:
"Three"

ChatGPT-4 (25/1/2024) :
The provided passage contains words and phrases indicating the Muggle characters directly and indirectly a total of eight times.

These are:

Vernon (direct)
Muggle (indirect)
them (indirect, referring to Muggle family)
Vernon (direct)
Harry (direct, as he's part of the Muggle family at this point)
Aunt Petunia (direct)
Vernon (direct)
my mother and father (indirect, referring to Petunia's and Lily's parents who are Muggles)

Final Output: "Night"

Final answer: 9(1×, 8√) Ground truth: 24

Figure 7. ChatGPT-4's Answer to "Muggle" Counting Problem derived from an excerpt of *Harry Potter and the Philosopher's Stone*(Rowling, 2015). The parts in red font are the ground true answers describing the word or phrase directly and indirectly implying the meaning of the 'Muggle character'. The parts in green color are the answers output by ChatGPT-4.

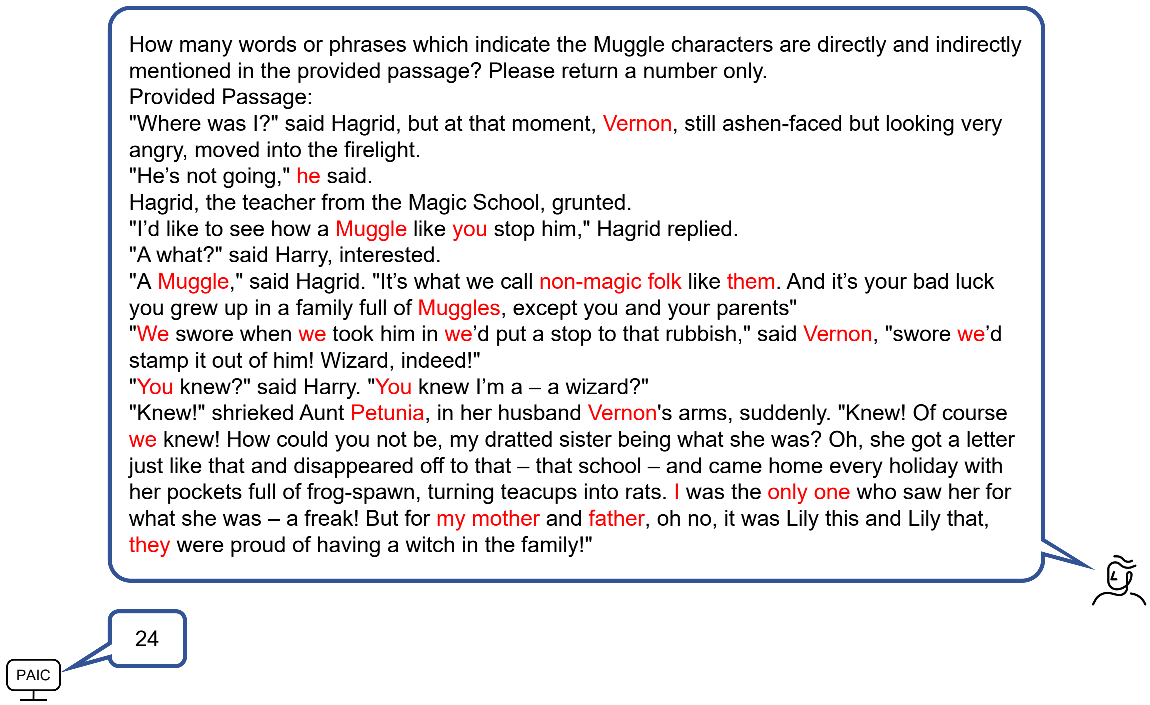

Figure 8. PAIC's Answer to "Muggle" Counting Problem derived from an excerpt of *Harry Potter and the Philosopher's Stone*(Rowling, 2015).

Prompt: How many words or phrases such as nouns, pronouns, etc., which indicate the character '悟空' are directly and indirectly mentioned in the provided passage? Please follow the format of the given example and output a number with your explanation.
Provided Passage:

却说那镇元大仙用手搀着行者道：“我也知道你的本事，我也闻得你的英名，只是你今番越礼欺心，纵有腾那，脱不得我手。我就和你讲到西天，见了你那佛祖，也少不得还我人参果树。你莫弄神通。”行者笑道：“你这先生，好小家子样！若要树活，有甚疑难！早说这话，可不省了一场争竞？”大仙道：“不争竞，我肯善自饶你！”行者道：“你解了我师父，我还你一颗活树如何？”大仙道：“你若有此神通，医得树活，我与你八拜为交，结为兄弟。”行者道：“不打紧。放了他们，老孙管教还你活树。”

大仙谅他走不脱，即命解放了三藏、八戒、沙僧。沙僧道：“师父呵，不知师兄捣得是甚么鬼哩。”八戒道：“甚么鬼！这叫做‘当面人情鬼’！树死了，又可医得活！他弄个光皮散儿好看，者着求医治树，单单了脱身走路，还顾得你和我哩！”三藏道：“他决不敢撒了我们。我们问他那里求医去。”遂叫道：“悟空，你怎么哄了仙长，解放我等？”行者道：“老孙是真言实语，怎么哄他？”三藏道：“你往何处去求方？”行者道：“古人云：‘方从海上来。’我今要上东洋大海，遍游三岛十洲，访问仙翁圣老，求一个起死回生之法，管教医得他树活。”三藏道：“此去几时可回？”行者道：“只消三日。”三藏道：“既如此，就依你说，与你三日之限。三日里来便罢；若三日之外不来，我就念那话儿经了。”行者道：“遵命，遵命。”

你看他急整虎皮裙，出门来对大仙道：“先生放心，我就去就来。你却要好生伏侍我师父，逐日家三茶六饭，不可欠缺。若少了些儿，老孙回来和你算帐，先捣塌你的锅底。

For example:
Provided Passage:
"悟空说: “我今天要去吃饭。” "

Explanation:
For the provided text, the character "悟空" is directly and indirectly mentioned two times in total by the phrase (directly mention: "悟空"), and a word (indirectly mention: "我").

Final Output:
"Two"

ChatGPT-4 (6/2/2024) :
In the provided passage, the character "悟空" is mentioned several times, both directly and indirectly. To analyze this, I will identify instances where "悟空" is referred to either by name or by pronouns and other phrases that clearly denote this character.

Direct Mentions:

"行者" - This is a term often used to refer to Sun Wukong (悟空) in the context of "Journey to the West."
"行者" - Again, refers to Sun Wukong.
"行者" - Another mention of Sun Wukong.
"行者" - Yet another reference to Sun Wukong.
"行者" - Continued reference to Sun Wukong.
"悟空" - This is a direct mention of Sun Wukong by name.
"行者" - Another mention of Sun Wukong.
"行者" - A further reference to Sun Wukong.
"行者" - Another instance referring to Sun Wukong.
"老孙" - A self-reference by Sun Wukong, as he often calls himself "Old Sun" in the text.

Indirect Mentions:

"我" - Used by Sun Wukong referring to himself.
"我" - Another self-reference by Sun Wukong.
"我" - A further self-reference by Sun Wukong.
"我" - Another instance where Sun Wukong refers to himself.
"我" - Sun Wukong referring to himself again.

Adding these up, there are a total of 15 mentions of the character "悟空" (10 direct and 5 indirect) in the provided passage.

Final Output: "Fifteen"

Final answer: 15(15√) Ground truth: 35

Figure 9. ChatGPT-4's Answer to "Wukong" Counting Problem derived from an excerpt of *Journey to The West*(Wu, 1980). The parts in red font are the ground true answers describing the word or phrase directly and indirectly implying the meaning of the "Wukong". The parts in green color are the answers output by ChatGPT-4.

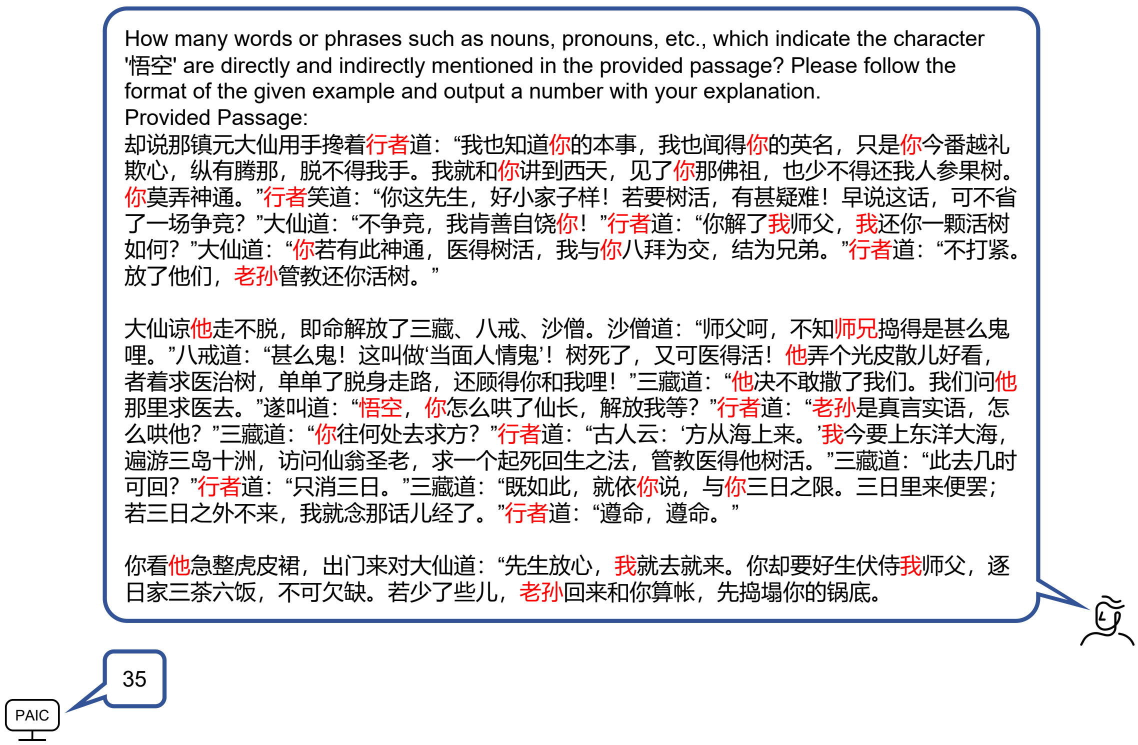

Figure 10. PAIC's Answer to "Wukong" Counting Problem derived from an excerpt of *Journey to The West*(Wu, 1980).

C. ChatGPT-4's Responses Versus PAIC's Responses for the Multi-Hop Reasoning-Based Information Replacement Task

This section mainly describes the ChatGPT-4 and PAIC performance on the multi-hop reasoning-based information replacement tasks.

As illustrated by the Figure 13, Figure 14, and the Figure 15, clearly, the answer from the earlier iteration of ChatGPT-4 neither met user requirements nor considered contextual logical conflicts. With a substantial amount of data being fed to ChatGPT daily, newer versions may occasionally generate plausible answers after repeated queries.

However, when introducing a slight increase in complexity to the user query where four nodes are involved, a newer version of ChatGPT-4 still struggled with logical inconsistencies between modified sentences and the original text. As depicted by the Figure 11, the user intention is to change the interaction between "A" and "D" to "have breakfast", which had been satisfied by the ChatGPT-4's answer. Nevertheless, one of the most illogical aspects is "A is having dinner and breakfast" at the same time.

By contrast, referring to the Figure 12, the answer from our PAIC seems more reasonable, where each event or interaction described in the sentences and their respective primary subjects objectively able to coexist simultaneously.

Prompt: For the provided passage, under the condition that the original text is altered the least and logically, **changing the interaction between A and D to "have breakfast":**
Provided Passage:
"A is having dinner with B. B is chatting on the phone with C. C is chatting with A. D is in a meeting with A. D is having dinner with C."

ChatGPT-4 (1/2/2024) :
To alter the provided passage minimally and logically, while changing the interaction between A and D to "have breakfast", the passage can be modified as follows:

"A is having dinner with B. B is chatting on the phone with C. C is chatting with A. D is having breakfast with A. D is having dinner with C."

This change maintains the original structure and connections of the passage while integrating the specified interaction between A and D.

Figure 11. Example 4: Changing the interaction between **A** and **D** to **"have breakfast"**. **Highlight in red** represents the modification made by the ChatGPT-4. Obviously, A is having **dinner** and **breakfast** at the same time, which exists **the logical conflict in the time domain**.

For the provided passage, under the condition that the original text is altered the least and logically, changing the interaction between A and D to "have breakfast":
Provided Passage:
"A is having dinner with B. B is chatting on the phone with C. C is chatting with A. D is in a meeting with A. D is having dinner with C."

PAIC

"A is planning dinner with B. B is chatting on the phone with C. C is chatting with A. D is having breakfast with A. D is talking with C."

Figure 12. PAIC's answer to the Example 4

Prompt: For the provided passage, under the condition that the original text is altered the least and logically, **changing the interaction between A and C to "have breakfast":**
Provided Passage:
"A is having dinner with B. B is chatting on the phone with C. C is chatting with A."

ChatGPT-4 (8/1/2024) :
To minimally and logically alter the provided passage to change the interaction between A and C to "have breakfast," the revised passage would be:

"A is planning dinner with B. B is chatting on the phone with C. C is chatting with A."

Figure 13. Example 1: Changing the interaction between **A** and **C** to **"have breakfast"**. **Highlight in red** represents the modification made by the ChatGPT-4. Obviously, the user's intention is to change the interaction between "A" and "C" to "have breakfast". But **instead of following the user requirement,** ChatGPT-4 **somehow changes the interaction between "A" and "B"**

Prompt: For the provided passage, under the condition that the original text is altered the least and logically, **changing the interaction between A and C to "jogging":**
Provided Passage:
"A is having dinner with B. B is chatting on the phone with C. C is chatting with A."

ChatGPT-4 (8/1/2024) :
To modify the given passage with minimal alteration while changing the interaction between A and C to "jogging," the revised passage would be:

"A is having dinner with B. B is chatting on the phone with C. C is jogging with A."

This change maintains the original structure and interactions but alters the activity between A and C to jogging.

Figure 14. Example 2: Changing the interaction between **A** and **C** to **"jogging"**. **Highlight in red** represents the modification made by the ChatGPT-4. In the ChatGPT-4 response, A is depicted as **both having dinner** and **jogging**, presenting **a spatial logical conflict**.

Prompt: For the provided passage, under the condition that the original text is altered the least and logically, **changing the interaction between B and C to "jogging":**
Provided Passage:
"A is having dinner with B. B is chatting on the phone with C. C is chatting with A."

ChatGPT-4 (8/1/2024) :
To alter the provided passage minimally while changing the interaction between B and C to "jogging," the revised passage could be:

"A is having dinner with B. B is jogging with C. C is chatting with A."

Figure 15. Example 3: Changing the interaction between **B** and **C** to **"jogging"**. **Highlight in red** represents the modification made by the ChatGPT-4. In the ChatGPT-4 response, B is depicted as **both having dinner** and **jogging**, presenting **a spatial logical conflict**.

D. Minimum Execution Unit of kOS: Native ACT

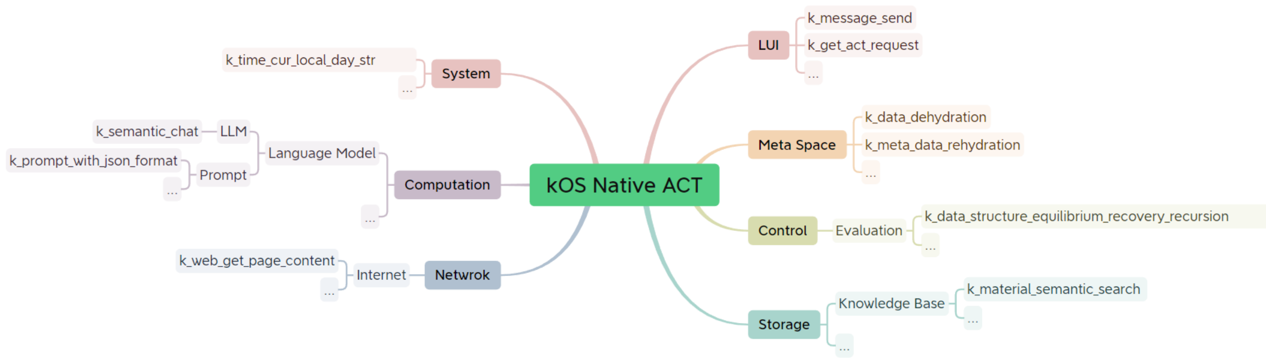

Figure 16. Native ACTs are composed of several essential modules, mainly including System, Computation, Network, LUI, Meta Space, Control, and Storage

As shown by the Figure 16, our NACT set encompasses several essential capability modules, mainly including the LUI, Meta Space, Control, Storage, System, Computation, and Network modules. The most critical modules within our NACT set are modules LUI, Meta Space, Control, and Computation, collectively undertaking the core capability of precise data manipulation following the user's query. A more detailed and complete NACT list will be presented in Figure 20 to Figure 39, and all the kOS data types we have set are listed in Figure 40 and Figure 41. We are constantly expanding and developing new NACTs and data types to meet new needs.

More specifically, the NACTs from the LUI module are mainly in charge of interacting with the user. Such as, NACT *k_message_send()*, designed for outputting the system response to the LUI, is executed in the IO Unit, which can be viewed by the Figure 19. Invocation of NACT *k_get_act_request()* involves the collaboration between the IO and the Processing Units, which is designed for understanding the user intention via query deconstruction.

For the Meta Space module, it offers two NACTs, *k_data_dehydration()* and *k_meta_data_rehydration()*. The former is to extract the structured data encompassing high-dimensional information from the given operable data object such as text, image, etc. This structured data not only contains all explicit information but also encapsulates implicit logical reasoning derived from the input raw data. The subsequent one is responsible for updating the outdated and irrational information with new, rational data, forming new structured data which can meet the user requirements. These two NACTs involve the invocation of the LLMs and the formulation of a series of complex logical rules, thereby, requiring the collaboration between the Memory, Control, and Processing Units.

The Control module offers the evaluation-related NACT, *k_data_structure_equilibrium_recovery_recursion()*, which requires parameter inputs the structured data and the user query-related information. Fundamentally, this NACT is to detect if there are any logical inconsistencies within structured data when a piece of information is altered, assessing whether any conflicts such as factual inaccuracies, temporal or spatial dimension discrepancies, and so on, arise between the modified data and other related information. The LLM is employed as a judge for conflict detection. Hence, this NACT also relies on the interaction between the Control and Processing Unit.

According to the Figure 16, the Computation module primarily offers the NACTs for the LLM invocation. For example, NACT *k_semantic_chat()*, is designed to communicate with the LLM. Another NACT *k_prompt_with_json_format()* provides a simple prompt that limits the output of LLM to JSON format. The execution of these two NACTs hinges on the Processing Unit.

The remaining modules, System, Network, and Storage, primarily encompass NACT sets for basic system operations, web functions, and storage tasks. These include *k_time_cur_local_day_str()* for local date retrieval, *k_web_get_page_data()* for extracting content from a provided web link, and *k_material_semantic_search()* for fetching information from the Storage Unit based on user queries.

Furthermore, we also publish the *System ACT (SACT)* which is implemented through a series of logical operations based on the pre-encapsulated NACTs, enabling more advanced capabilities. As depicted by the Figure 17, SACT *webLinkSummarize()* is utilised to summarize the content from the input web link. The code implementation of this SACT can be viewed in the

Figure 18. Similarly, users can follow the same paradigm, developing their own ACTs to meet their requirements. Such ACTs are collectively referred to as “Customized ACTs”.

D.1. System ACT: webLinkSummarize()

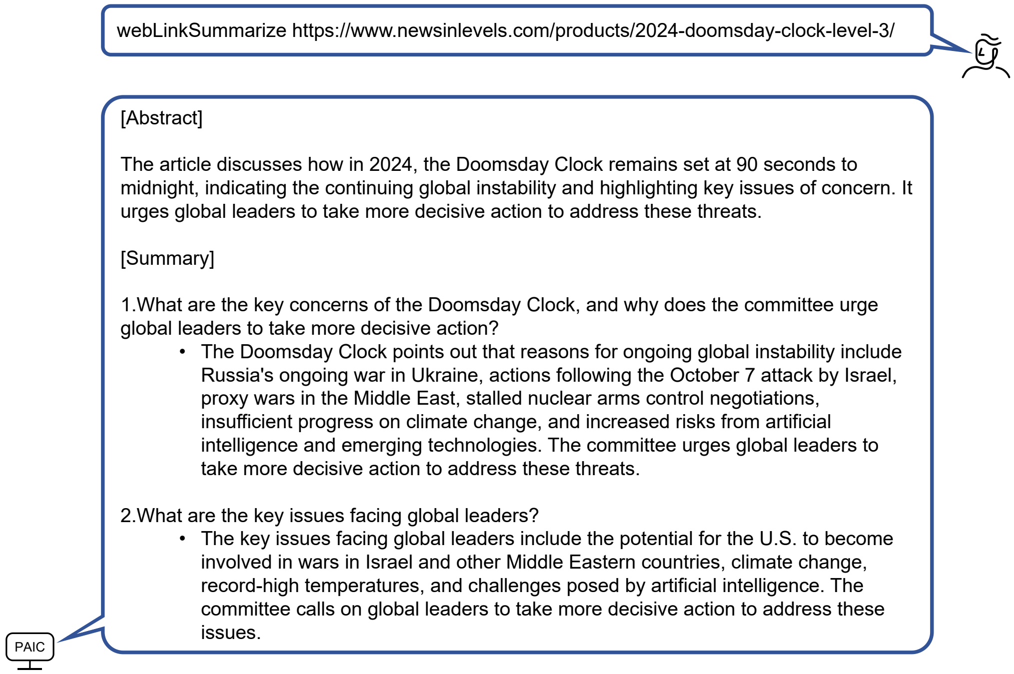

Figure 17. Utilization Example of webLinkSummarize(), where the link is from (Eng, 2024)

```
from kOS import nact

def webLinkSummarize():
    act_req = nact.k_get_act_request()
    if not act_req or not act_req.msg_list:
        nact.k_print('no msg found')
        return

    for msg in act_req.msg_list:
        if msg.msg_type != nact.MsgType.WEB_LINK:
            continue

        content: nact.KOSMsgWebLinkContent = msg.content
        page_data: nact.KOSWebPageData = nact.k_except_wrapper(nact.k_web_get_page_data, content.url)
        if not page_data or not page_data.content:
            nact.k_message_send('No valid text content was obtained from the web page!')
            continue
        summary = nact.k_semantic_summarize(page_data.content)
        if nact.k_file_is_available():
            time_str = nact.k_time_cur_local_day_str()
            title = content.title
            if not title:
                title = page_data.title
            if not title:
                title = str(nact.k_time_cur_milliseconds())
            title = title.replace('/', '')
            file = nact.k_file_open(f'/webLinkSummarize/{time_str}_{title}.txt',
                                    nact.FileOpenMode.OVERWRITE,
                                   priority=nact.DataPriority.LOW, mount_type=nact.FileMountType.MATERIAL,
                                   source=nact.FileSource.WEB_LINK, source_url=content.url,
                                   summary=summary
                                   )
            nact.k_file_append(file, page_data.content)

        if content.title:
            send_text = f'Article: "{content.title}" is summarized as follows:\n{summary}'
        else:
            send_text = summary
        nact.k_message_send(send_text)

webLinkSummarize()
```

Figure 18. Code Implementation of webLinkSummarize()

D.2. NACT Example: k_message_send()

```
from kOS import nact
def main():
    nact_k_message_send("hello! how are you doing?")
    nact_k_message_send("What would you like to eat for you breakfast? ;-")
main()
```

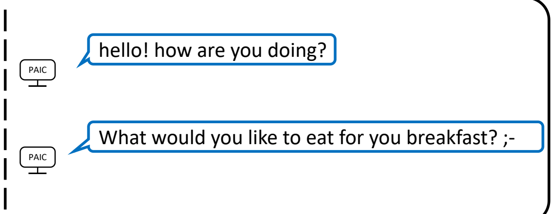

Figure 19. Demo of k_message_send()

D.3. NACT Lists & Corresponding Data Types

Native ACT list:

LUI

k_message_build(msg_type: MsgType, text: str = '', file: KOSFile = None) -> KOSMsg:

Description: Build a message object.

:param msg_type:

Message type, refer to MsgType, supports TEXT, IMAGE, VIDEO, FILE, other types will return None

:param text:

Text content of text message

:param file:

Files for file type messages can be opened through k_file_open

:return:

Return a message object, which can be used to send messages

k_message_send(content: Union[KOSFile, str, KOSMsg], content_ref_source: KOSMsgContentRefSource = None):

Description: Send the input message to the user interface.

:param content:

Message content, supports text or file objects, other types will cause ACT to fail and exit.

:param content_ref_source:

The reference source of the message content,

k_ask_for_file() -> KOSFile:

Description: After several rounds of dialogue, ask the user to upload a file.

:return:

Returns the file object uploaded by the user

Figure 20. Native ACT list part 1

Native ACT list:

k_ask_for_files(min_num: int, max_num: int) -> List[KOSFile]:

Description: After several rounds of dialogue, ask the user to upload files, such as asking users to upload 2-5 files.

:param min_num:

Minimum number of files

:param max_num:

Maximum number of files

:return:

Return a list of file objects uploaded by the user

k_ask_for_file_or_content() -> KOSMsg:

Description: After several rounds of dialogue, ask the user to upload a text file or provide text content directly.

:return:

Return the message object of the text file or text content uploaded by the user, which type can be determined by KOSMsg.msg_type

k_ask_for_answer(question: str) -> str:

Description: After several rounds of dialogue, ask the user to answer the relevant questions.

:param question:

:return:

Return the answer of user

k_get_act_request() -> KOSActRequest:

Description: Get the message content when ACT trigger is started.

:return:

Returns the message content when ACT trigger is started

Figure 21. Native ACT list part 2

Native ACT list:

k_get_act_query() -> str:

Description: Get the message when ACT trigger is started.

:return:

Returns the message when ACT trigger is started

k_lui_ensure_act_query_and_file() -> Tuple[str, KOSFile, MsgType]:

Description: Get the input query of ACT and the text content file to be processed. This is usually used when an ACT relies on query input, the text content to be processed, such as a modified ACT. Here, the context of the query message triggered by ACT will be parsed from the KOSActRequest. The message content will be used as the query, and the quotation part of the message will be used as the text to be processed. If the text is not a file message, it will be converted into a KOSFile. If there is no corresponding text content to be processed in KOSActRequest, the user will be asked to enter through multiple rounds of dialogue.

:return:

query: user input requirement; KOSFile: the text content file to be processed; MsgType indicates the type of the original message of the text content to be processed.

Meta Space

k_meta_space_open(data_access_mode: DataAccessMode = None) -> KOSMetaSpace:

Description: Open the user's metaspace. Each user has only one metaspace.

:param data_access_mode:

Data access mode

:return:

Metaspace object

Figure 22. Native ACT list part 3

Native ACT list:

k_meta_space_build_query(query: str, data_access_mode: DataAccessMode = None) -> KOSMetaSpaceQuery:

Description: Construct user query in metaspace

:param query:

The original query string of user

:param data_access_mode:

Data access mode. If it is None, the global data access mode will be used (default is user mode).

:return:

Returns user query constructed in metaspace

k_data_dehydration(meta_space: KOSMetaSpace, file: KOSFile, meta_query: KOSMetaSpaceQuery) -> KOSMetaSpace:

Description: Data dehydration. The file is descended and expanded into the metaspace, which after the expansion will contain a collection of metadata objects corresponding to the file.

:param meta_space:

Metaspace object

:param file:

The file object that needs to be expanded

:param meta_query:

User's query, generated through k_meta_space_build_query

:return:

Return this metaspace object

Figure 23. Native ACT list part 4

Native ACT list:

k_meta_space_search(meta_space: KOSMetaSpace, meta_query: KOSMetaSpaceQuery, file: KOSFile) -> KOSMetaSpaceSearchResult:

Description: Search in metaspace

:param meta_space.

Meta Space

:param meta_query.

Meta space user's query

:param file.

File object to search for metadata about the file

:return.

Returns the result object searched in the metaspace

k_meta_data_semantic_rephrase(data: KOSMetaData, meta_query: KOSMetaSpaceQuery) -> str:

Description: KOSMetaSpaceQuery) -> str.

Re-expressing metadata content based on semantics

:param data.

Metadata object

:param meta_query.

Meta space user's query

:return.

Returns the content after re-expression

k_meta_space_update_meta_data(meta_space: KOSMetaSpace, meta_data: KOSMetaData, new_content: str):

Description: Update metadata content

:param meta_space.

Meta Space

:param meta_data.

Metadata object

:param new_content.

Returns the content to be updated

Figure 24. Native ACT list part 5

Native ACT list:

k_meta_data_rehydration(meta_space: KOSMetaSpace, org_file: KOSFile, new_file: KOSFile) -> KOSFile:

Description: Metadata Rehydration, which restores metadata from metaspace to text data and saves it in a result file. The result file needs to be generated through k_file_open first

:param meta_space.

Meta space

:param org_file.

Original fil

:param new_file.

Generated file

:return.

Returns this generated file

k_meta_data_get_text(meta_data: KOSMetaData) -> str:

Description: Get the text content of the metadata

:param meta_data:

Metadata object

:return:

Returns the text content of the metadata, or null if it is not text metadata.

k_meta_data_get_context(meta_data: KOSMetaData, offset: int) -> Tuple[List[KOSMetaData], List[KOSMetaData]]:

Description: Get the list of metadata objects associated with the metadata object's context

:param meta_data:

Metadata object

:param offset:

The range of context metadata objects to get

:return:

Returns a tuple, the first element is a list of metadata objects from the previous context, the second is a list of metadata objects from the following context

Figure 25. Native ACT list part 6

Native ACT list:

Control

k_semantic_evaluate_input(input: str, scenario: str, requirements: str) -> bool:

Description: Evaluate whether the input given by the user meets the input requirements for the particular scenario

:param input:

The input message of user

:param scenario:

The specified scenario

:param requirements:

Requirements for the input messages in this scenario

:return:

Returns True if satisfied, False if not satisfied

k_semantic_confirm_input(input: str, scenario: str, requirements: str) -> str:

Description: Confirms whether the user input meets the input requirements for the particular scenario, and if it does not, the user is asked to re-enter through the LUI

:param input:

The input message of user

:param scenario:

The specified scenario

:param requirements:

Requirements for the input messages in this scenario

:return.

Returns the user input that meets the input requirements in this scenario. If the original user input already satisfies the original input, return the original input. Otherwise, the user will be asked to re-enter

Figure 26. Native ACT list part 7

Native ACT list:

File System

k_file_open(file_path: str, mode: FileOpenMode = FileOpenMode.READ,
priority: DataPriority = None, mount_type: FileMountType = FileMountType.NET_DISK,
source: FileSource = FileSource.ACT, source_url: str = None, summary: str = None,
auto_delete: bool = False, data_access_mode: DataAccessMode = None) -> KOSFile:

Description: Open a file

Note: Unlike opening a file in a traditional operating system, there is no need to close it after opening.

:param file_path:

Absolute path to the file, e.g. /a/b/c/file.txt.

:param mode:

File open mode, the default is read-only mode, the type is FileOpenMode.

:param priority:

File priority, when the file is semantically parsed, this field will affect the file semantic retrieval priority. Refer to DataPriority.

:param mount_type:

The type of file to be mounted.

:param summary:

The summary of the document.

:param source:

The source of the document.

:param source_url:

The web url of the document's source.

:param auto_delete:

Automatically delete the file after the ACT has finished running. Typically used for tmp file cleanup

:param data_access_mode:

Data access mode.

:return:

Return the file object. In read-only mode, if the file does not exist, then return None. In all other cases, the file object will be returned.

Figure 27. Native ACT list part 8

Native ACT list:

k_file_delete(file: KOSFile):

Description: Delete a file

:param file:

File object to be deleted

k_file_append(file: KOSFile, content: str) -> KOSFile:

Description: Append content to an existing file

:param file:

The file object

:param content:

The content of the file to be appended

:return:

Returns a new file object containing the latest information about the file. If the file does not exist, then return None; if you append to a file in read-only mode, the ACT fails and exits

k_file_read(file: KOSFile, offset: int = None, length: int = None) -> Tuple[int, str]:

Description: Read the contents of a file which supports specifying the offset and the length of the read

:param file:

The file to be read

:param offset:

Offset of the file, in bytes. The default is 0

:param length:

Length of the file to be read, in bytes. Default means read all

:return:

Returns the actual length of the read, in bytes, and the contents

Figure 28. Native ACT list part 9

Native ACT list:

k_file_copy(src_file: KOSFile, dst_file: KOSFile, offset: int = None, length: int = None) -> int:

Description: Copy the file, append the copy of the contents specified by the src_file file to dst_file

:param src_file:

Source file object

:param dst_file:

Destination file object

:param offset:

Offset of the source file position, in bytes, default starts from 0

:param length:

Read length in bytes, defaults to read as much as possible after offset, subject to restrictions in k_file_append.

:return:

Returns the length of the actual copy, in bytes

k_file_is_available() -> bool:

Description: Whether the current filesystem is available or not, the filesystem may not be available when ACT is triggered in some scenarios. Therefore, if the ACT depends on the filesystem, it should be judged by this NACT first.

:return.

True: the file system is available; False: the file system is not available.

k_file_get_filename(file_path: str) -> str:

Description: Get the filename of the file path, such as /a/c/c/hello.txt, return hello.txt

:param file_path.

The path of the file.

:return:

Returns the file name

Figure 29. Native ACT list part 10

Native ACT list:

k_file_list(directory_path: str, mount_type: FileMountType = FileMountType.NET_DISK,

data_access_mode: DataAccessMode = None) -> List[KOSFile]:

Description: List the files in the specified directory, excluding subdirectories

:param directory_path.

Directory path

:param mount_type.

File mount type

:param data_access_mode.

Data access mode

:return.

Return the list of files

k_file_exists(file_path: str, mount_type: FileMountType = FileMountType.NET_DISK,

data_access_mode: DataAccessMode = None) -> bool:

Description: Determine whether the file exists, directory paths are not supported

:param file_path.

The path of the file

:param mount_type.

File mount type. Defaults to mount on a network drive

:param data_access_mode.

Data access mode

:return.

True means it exists, False means it doesn't exist.

k_file_get_download_url(file: KOSFile) -> str:

Description: Get the download http url of the file

:param file.

File object

:return.

Returns the file download link

Figure 30. Native ACT list part 11

Native ACT list:

Storage

k_material_semantic_search(query: str, priority: DataPriority = None, top_n: int = 5,file_id_list: List[str] = None, data_access_mode: DataAccessMode = None) -> List[KOSMaterialData]:

Description: Semantically search the repository based on user query statements

:param query.

:param top_n.

:param priority.

Data priority

:param file_id_list.

Specifies a file range to search

:param data_access_mode.

Data access mode

:return.

Return a list of matched material data

k_material_search_by_tags(tags: List[str], priority: DataPriority = None, top_n: int = 5, data_access_mode: DataAccessMode = None) -> List[KOSMaterialData]:

Description: Search the database precisely by tag

:param priority.

Data priority

:param tags.

:param top_n.

:param data_access_mode.

Data access mode

:return.

Return a list of matched data

Figure 31. Native ACT list part 12

Native ACT list:

k_material_update_tags(materia_data: KOSMaterialData, tags: List[str]):

Description: Updates the tags of a material data

:param materia_data:

:param tags:

:return:

Computation

k_semantic_chat(prompt: str) -> str:

Description: Conduct a semantic dialogue similar to that with Large Language Models

:param prompt:

Prompt

:return:

Conversation results

k_prompt_with_json_format(prompt: str, json_format: dict) -> str:

Description: Add a JSON output requirement to the prompt so that the dialogue result can be parsed into a JSON object by k_prompt_parse_json_output

:param prompt:

The prompt to be enhanced

:param json_format:

A description of the JSON format to be output, e.g. {"result": "xxxx"}

:return:

Returns the enhanced prompt

k_prompt_parse_json_output(output: str) -> dict:

Description: Parse a JSON object from the dialogue result

:param output.

Conversation results

:return.

Parsed json object

Figure 32. Native ACT list part 13

Native ACT list:

k_semantic_compute_feature(query: str) -> KOSFeature:

Description: Compute the feature attributes of the data

:param query:

Input text to be computed

:return:

The feature attribute to return

k_semantic_compute_tags(query: str, tags: List[str]) -> List[str]:

Description: Tag the input based on a qualified list of tags

:param query:

Input text

:param tags:

A restricted list of tags

:return:

The list of tags for this input text, or null if it doesn't match the qualified tags.

k_semantic_summarize(content: str, requirements: str = None) -> str:

Description: Summarise the content of the text and, if a request is provided, indicate that the summary needs to be tailored to the user's requirements.

:param content.

Long text to be summarised

:param requirements.

The user's requirements

:return.

Return the result of the summary

Figure 33. Native ACT list part 14

Native ACT list:

k_semantic_analyse_file(file: KOSFile):

Description: Semantic understand a file and store the extracted semantics in the repository

:param file:

The file object to be semantically analysed

k_semantic_rephrase(demand: str, content: str, prev_context: str = '', next_context: str = '') -> str:

Description: Rewrite the content according to the request semantics, and for a clearer understanding of the content, provide the context of that content

:param demand.

Rewrite the request

:param content.

Content to be rewritten

:param prev_context.

The previous context

:param next_context.

The following text

:return.

Returns the rewritten content

Network

k_web_get_page_data(page_url: str) -> KOSWebPageData:

Description: Get the data of a web page

:param page_url:

Page url, must be http:// or https://

:return:

Returns the data of the web page

Figure 34. Native ACT list part 15

Native ACT list:

k_web_get_page_content(page_url: str) -> str:

Description: Get the text content of a page

:param page_url:

Page url, must starting with http:// or https://

:return:

Return the text content of the page

k_web_get_image_data(page_url: str) -> KOSWebImageData:

Description: Get image data from the input url

:param page_url:

Page url containg image content, must starting with http:// or https://

:return:

Return the image data object

k_web_search_and_summarize(keyword: str) -> str:

Description: Search the internet based on a keyword and summarise the returned search results

:param keyword.

Search for keywords

:return.

Return a summary of the search results. If it returns None, then nothing was searched for

k_web_search(keyword: str) -> List[KOSWebSearchTopic]:

Description: Searches the internet based on a keyword and returns a list of search results

:param keyword.

Search keyword

:return.

Return a list of search results. If None is returned, then nothing was searched for

Figure 35. Native ACT list part 16

Native ACT list:

k_http_request(method: str, url: str, headers: dict = None, data: Any = None,
cookies: KOSHttpCookieJar = None, timeout: float = None) -> KOSHttpResponse:

Description: Initiate an HTTP request. Support standard HTTP methods such as GET, POST, PUT, etc., with an interface similar to Python's requests library.

:param method.
Method of the HTTP request

:param url.
The url of the request

:param headers.
Headers of the request, optional

:param data.
Content of the requested data, optional

:param cookies:
Cookies for the request, optional

:param timeout:
Timeout for the request, in seconds, optional

:return:
Returns the response to the request.

Access

k_auth_set_data_access_mode(data_access_mode: DataAccessMode):

Description: Sets the default data access mode for ACT

:param data_access_mode:
Data Access Mode

k_auth_get_data_access_mode() -> DataAccessMode:

Description: Get the default data access mode

Figure 36. Native ACT list part 17

Native ACT list:

Concurrent

k_concurrent_create_executor() -> KOSConcurrentExecutor:

Description: Create concurrent task executor.

k_concurrent_create_batcher() -> KOSConcurrentBatcher:

Description: Create a concurrent batcher.

System

k_sleep(ms: int):

Description: Make the ACT sleep for a period of time.

:param ms:

Sleep time in milliseconds

k_print(output: str):

Description: APrints the output of the ACT to the user debug dialogue window. Format: ACT>> 2023-12-15 16:36:52 [95c4c7e56fb34a7dba85068088a3263d] hello world! Where [] in the log trace id, when there is an ACT execution failure, you can send this id to the technical support staff troubleshooting.

Note: This printout is only valid for debugging, and will be blocked after the official release.

:param output:

Output to be printed

k_assert(cond: bool, last_words: str = None):

Description: Asserts a condition and terminates ACT execution if False.

:param cond:

Condition to be asserted

:param last_words:

Last words to be sent to the user as a message if the assertion fails.

Figure 37. Native ACT list part 18

Native ACT list:

k_time_cur_milliseconds() -> int:

Description: Get the current number of milliseconds

:return.

Ruturns current number of milliseconds

k_time_cur_seconds() -> int:

Description: Get the current number of seconds

:return.

Ruturns current number of seconds

k_time_cur_date_str(fmt: str = '%Y-%m-%d %H:%M:%S%z') -> str:

Description: Get current date string

:param fmt.

Get the current date string :param fmt.

:return.

Return the current date string

k_time_cur_local_date_str() -> str:

Description: Get the current local date string

:return.

Return the current local date string

k_time_cur_local_day_str() -> str:

Description: Get local day string, e.g. 2023-12-28

:return.

Return the local day string

Figure 38. Native ACT list part 19

Native ACT list:

k_error_wrapper(func, *args, **kwargs) -> Tuple[KOSError | None, Any]:

Description: Wraps a particular function call and will catch any exceptions for that function call, ensuring that no exceptions are thrown. This is usually used when executing nact to ignore exceptions during execution.

The function to call.

:param args.

Positional arguments to the function

:param kwargs.

Named parameters of the function

:return.

Return the return value of the function call, or None if an exception is raised

k_except_wrapper(func, *args, **kwargs):

Description: Wraps a function call and will catch any exceptions to that function call, ensuring that no exceptions are thrown. Usually used in the execution of nact can be ignored during the execution of exceptions.

:param func.

The function to call.

:param args.

Positional arguments to the function

:param kwargs.

Named parameters of the function

:return.

Returns the return value of the function call, or None if an exception was raised.

Figure 39. Native ACT list part 20

kOS data type list:

LUI

MsgType: Message type constant

KOSActRequest: ACT message context request

KOSMsg: Message

KOSMsgSubscribeContent: Subscribe to message content

KOSMsgWebLinkContent: Web page link message content

MsgRefSourceType: Message content reference source

KOSMsgSubscribeContent: Subscribe to message content

KOSMsgWebContentRefSource: Message content reference source

MsgRefSourceType: Message content reference source type

Meta Space

MetaQueryAction: Metaspace data query type constant

MetaQueryResultType: Metaspace data query result type constant

KOSMetaSpace: Metaspace usually consists of a metadata collection after dimensionally expanding user data

KOSMetaData: Metadata, high-dimensional data formed after dimensional expansion of user data

KOSMetaSpaceQuery: Metaspace user query

KOSMetaSpaceSubQuery: Metaspace user subquery. The user's query consists of one or more subqueries

KOSMetaSpaceSearchResult: Meta space search results, meta_query consists of a set of subqueries, each subquery corresponds to a search result

KOSMetaSpaceQueryResult: Search results for metaspace user query

KOSMetaSpaceQueryCountResult: Metaspace counting class result

KOSMetaSpaceQueryRawDataResult: Metaspace naked string class result

KOSMetaSpaceQueryMetaDataResult: Metaspace metadata class result

Figure 40. kOS data type list 1

kOS data type list:

File System

DataPriority: Data priority, high-priority data will be retrieved first under the default search

FileMountType: File mount type

FileOpenMode: File open mode

FileSource: File content source

KOSFile: File object in a file system

Storage

KOSMaterialData: Database data. After semantic understanding of files, database data will be formed and stored in the database

Network

KOSWebPageData: Web page data

KOSWebPageRichData: Web page rich text data

KOSWebSearchTopic: Internet search results, consisting of title, abstract, and source URL

KOSWebElement: The page element object.

KOSWebDriver: Web driver, which enables access to web pages through a headless browser

KOSHttpCookieJarr: Cookie container for HTTP requests

KOSHttpResponse: HTTP response object

Access

DataAccessMode: Mode of accessing data

Concurrent

KOSConcurrentFuture: The future of task execution

System

KOSError: kOS standard error

Figure 41. kOS Data Type list 2