Preserving Secondary Knowledge - Using Language Models for Software Preservation

Klaus Rechert
University of Applied Sciences Kehl

Germany

Rafael Gieschke
University of Freiburg

Germany

Abstract

Emulation and migration are still our main tools for digital curation and preservation practice. Both strategies have been discussed extensively and have been demonstrated to be effective and applicable in various scenarios. Discussions have primarily centered on technical feasibility, workflow integration, and usability. However, there remains one important aspect when discussing these two techniques: managing and preserving operational knowledge. Both approaches require specialized knowledge but especially emulation requires future users to also have a great variety of knowledge about past software and computer systems for successful operation. We investigate how this knowledge can be stored and utilized, and to what extent it can be rendered machine-actionable, using modern large language models. We demonstrate a proof-of-concept implementation that operates an emulated software environment through natural language.

Submitted 9 February 2024 ~ *Accepted* 22 February 2024

Correspondence should be addressed to Klaus Rechert, University of Applied Sciences Kehl, Kinzigallee 1, 77694 Kehl, Germany. Email: rechert@hs-kehl.de

The *International Journal of Digital Curation* is an international journal committed to scholarly excellence and dedicated to the advancement of digital curation across a wide range of sectors. The IJDC is published by the University of Edinburgh on behalf of the Digital Curation Centre. ISSN: 1746-8256. URL: http://www.ijdc.net/

Introduction

The task of curating, managing, and preserving digital objects has always been challenging, and it continues to present new challenges constantly. In digital preservation practice, there are two primary approaches to ensure the availability, accessibility, and reuse of a digital object over time. The first approach focuses on the object's information content, aiming to continuously make it available in a contemporary representation by consistently updating the information's encoding to a modern format, typically achieved through format migration. The focus of this approach lies in the interpretation of byte streams (data). Each representation demands comprehensive documentation to facilitate its translation into a new, ideally equivalent, representation. This documentation is necessary when implementing migration software at the end of a file format's lifecycle. At this point in time, the availability of technical information is generally not a significant concern, with the notable exception of proprietary file formats. The documentation on formats as well as the migration process can also be preserved for validation, reproduction, or refinement of the process at a future point in time.

However, digital objects exist in numerous forms, making comprehensive application of format migration a complex endeavor due to this diversity. Additionally, software becomes an increasingly important preservation concern. Software is necessary to access information within complex object types that lack an available or acceptable migration path (e.g., digital art) or where migration is not an efficient option due to a very small number of objects. Thus, software is becoming a primary object in the preservation field, especially in research data management. In the DataPLANT project[1], the Annotated Research Context Specification[2] is being developed to describe research data and the software that has produced the data. It uses the Common Workflow Language (CWL) to document the used computational workflows, consisting of steps of running research software packaged as (Docker) container images. In contrast to format migration, it is one example of the alternative preservation route that focuses on preserving the software environment, e.g., the original software environment of a digital object (software or data), thereby enabling access and reuse. This method relies on maintaining the object's software stack and its technical and operational functionality, typically achieved through emulation. Since the so-called primary digital objects (data or software) remain unmodified, this method centers on gathering "secondary objects" such as software or source code. Building and maintaining the object's technological stack, as well as using the software environment to access the object, requires significant knowledge, e.g., knowledge of system operation and detailed understanding of operating a complex software product, both about the primary (software) object as well as the secondary objects (i.e., the software stack). While this knowledge exists within the contemporary user community and is usually well-documented and available through books, blog posts, or even YouTube videos, future users will also need this knowledge.

Hence, comparing migration and emulation with regard to the necessary and available knowledge as well as to the necessary preservation work to be carried out at different points in time presents an interesting trade-off. For instance, in the research data domain, among datasets, there is a high diversity of formats due to specialization and various (automated) data sources. A format-based preservation strategy requires documenting and monitoring the lifecycle of these formats and migrating each format if necessary. For fixed-term scientific projects, this usually means entrusting this task to a specialized institution due to the timing of format obsolescence and preparing as much information as possible to support this process, as long as resources and knowledge remain available. If information is missing, from that point on compiling this information will become increasingly challenging. Successfully migrated data

[1] https://nfdi4plants.org/
[2] https://github.com/nfdi4plants/ARC-specification/blob/main/ARC%20specification.md

usually eliminates concerns over its previous format. Hence, a migration strategy encapsulates necessary information and preservation actions within the same technological era.

In contrast, an emulation-based strategy may require the collection and documentation of today's runtime environment (secondary objects). In many cases, this task can be automated or even omitted, with some responsibility instead delegated to a generic software preservation strategy. This is because many primary objects need fairly generic runtime environments, or their runtime dependencies are shared among a large number of similar objects. Since the primary objects are maintained in their original form with none or minimal preservation work necessary today, the main workload is transferred to future users. These users will then run a future (emulated) software setup to access the primary object. Consequently, a significant amount of today's operational knowledge will be required to utilize these secondary objects in the future. However, even for software still in operation, it has been a long-known fact that users do not like to read documentation (Rettig, 1991). A solution should just provide help in a context-sensitive or, ideally, directly machine-actionable way. In other contexts, it has been shown that, e.g., chatbots can be successfully used for context-sensitive IT support (Fiore et al., 2020).

In this paper, we examine how this operational "secondary knowledge," necessary for accessing and effectively reusing digital objects, can be maintained and preserved. We investigate how this knowledge can be stored and utilized, and to what extent it can be rendered machine-actionable, using modern large language models. Lastly, we demonstrate a proof-of-concept implementation that operates an emulated software environment through natural language.

The Secondary Knowledge Dependency

If we assume that our ongoing efforts to preserve software and its documentation will be successful, future preservation administrators or end-users will face a vast collection of different software originating from various, then-obsolete computer systems. Each technical epoch features its own user-machine interaction paradigm, ranging from text-only interfaces to various kinds of mouse and keyboard-controlled graphical user interfaces, and now the emerging Augmented Reality/Virtual Reality systems. The same applies to (quasi-)standard operating software products.

Future users will be required to perform system operations, such as launching a program or searching for a file or content within a file. In a research data scenario, it might be necessary to manually set up, configure and operate a suitable software environment to access (and render) data. Specific software may be required and needs to be installed and configured within an obsolete operating system, such as OS/2 or an older Unix derivative. Additionally, in these cases, it is not uncommon for some software dependencies to be distributed (and therefore archived) as source code, which needs to be compiled on the target system. It is reasonable to assume that a future user may not have the requisite knowledge to carry out the tasks outlined and would therefore need to resort to documentation. Even if the software environment including dependencies and used parameters to launch, e.g., command-line programs, is preserved in a machine-actionable way, as is the case in preserved workflows in DataPLANT’s ARC packages, future users might want to alter these workflows in unforeseen ways, again requiring operational knowledge.

The required operational knowledge for such tasks is vast and will likely be available in the form of formal documentation, such as manuals and books, as well as in informal, anecdotal advice shared on platforms like Stack Overflow, Reddit, and other user forums. While formal documentation may provide the necessary technical details for setup and operation, informal sources often fill the gaps in understanding the quirks of complex computer systems. The amount of documentation to be processed could be substantial, depending on the user's prior knowledge and familiarity with the system in question. As software and systems age, it can be assumed that pre-existing knowledge will progressively diminish, making it increasingly difficult to articulate a problem statement—for instance, identifying and describing a specific system state. For example, today's commonly understood term "desktop," describing a specific state of the Windows operating system, might become less commonly known by future users. This situation is similar to the difficulties today's students face when trying to apply this concept (e.g., locating a file on the hard disk or CD-ROM) within a Windows for Workgroups 3.11 environment, which uses a vastly different desktop metaphor. Another example is the "any key" phenomenon[3].

A solution to this problem is to support future users in a context-aware manner. Specifically crafted and curated documentation, such as step-by-step guidelines for common tasks, can simplify the user's adaptation to an uncommon environment. Even though usability challenges remain, since the user is still required to translate the guidelines into actionable inputs using an unfamiliar human-machine interaction interface. To mitigate these problems, the step-by-step guidelines can also be made actionable (Oberhauser et al., 2022), such that the required input events are generated in an automated way. However, creating such comprehensive documentation requires dedicated effort and is typically tailored to a specific objective, such as performing specific system operations. Given that the documentation is available and presumably accessible to future users, the main challenge lies in transforming this information into operational, ideally machine-actionable knowledge.

Automating Access to Operational Knowledge

Although most of the required knowledge about old software and computer systems is available through formal or informal documentation, it is not easily accessible nor machine actionable. This difficulty arises because the knowledge is scattered across various sources and requires significant contextual and contemporary understanding to identify relevant documents and used terminology. A context-aware information retrieval system, which adapts to the environment such as the operating system and software being used, offers a promising solution. For instance, the user's current screen content could be analyzed and ideally, and with multiple observations even the user's intent could be identified. Based on the user's context, relevant pre-selected information could be prepared and actionable tasks be derived, either based on the detected user intent or the user's request.

LLMs as a Technical Knowledge Base

One way to implement such a context-aware information retrieval system would be to use a simple full-text search, for example, over a collection of indexed software manuals or even the entire Internet. It could constantly search for terms currently displayed on the user's screen and provide the search results to the user. The searched terms could be extended with the environment's static metadata, such as the computing platform used, operating system, or installed software. However, this approach might not be very helpful as it would still only show literal content from the software manuals. This content often requires further contextual and contemporary knowledge to be understood, may not be actionable by machines, and might not be adaptable for cases that are not directly described in the manual but have to be derived from information given in different sections of the manual. Additionally, it may not reliably find correct results at all, with search terms being too specific or too generic. For example, a dialog to change the text alignment of a paragraph in a word processor will contain the words "left" and "right," which will likely appear on many pages completely unrelated to the task at hand. In summary, software manuals only provide and full-text searches only use unstructured (non-actionable) data, lacking reasoning and derivation of combined knowledge. A big advantage of this approach, however, is that full-text search, especially over the whole Internet and enhanced with specific software manuals, provides access to a vast amount of knowledge and does not require special preparation for individual preserved software environments.

[3] Cultural significance or 'any key', https://en.wikipedia.org/wiki/Any_key#Cultural_significance

A traditional knowledge base (KB), on the other hand, can organize real-world knowledge in a much more structured form, typically describing facts as triples of subject, predicate, and object, in conjunction with an ontology. This allows for the description of relationships between facts. These systems are quite powerful in answering dedicated, formalized questions in a predictable and deterministic manner. A popular example of a general-purpose knowledge base is Wikidata (Vrandečić & Krötzsch, 2014). Alternatively, software manuals themselves can be described in a semantic way instead of only consisting of plain text (Loch & Stolze, 2023), facilitating their usage as a knowledge base. While the captured knowledge even in large projects such as Wikidata is only a fraction of the unstructured (full text) knowledge available to search engines indexing the whole Internet and (re)creating software manuals with semantic annotations is a laborious and non-scalable task, the main issue with these systems is the difficulty in translating real-world context into a formalized query and converting the result into an actionable response. For instance, a user may be looking at a screen of a running system that requires user input to proceed. The challenge lies in translating this situation into a formalized query and then translating the result into an actionable command to be entered.

Recently, large language models (LLMs) have transitioned from being a niche research field, known only to a select group of specialized researchers, to becoming almost a household name, popularly known as ChatGPT. This remarkable development is largely due to the significant advancements in their capabilities (Bubeck et al., 2023). Specifically, two properties of LLMs should also spark the interest of the data curation and archival community: providing a universal, natural language user interface and their ability to manage and provide access to a text-based, unstructured knowledge base. For this, LLMs are "pre-trained" on a huge amount of text data (e.g., Touvron et al., 2023) provide a rather detailed breakdown on training data used) usually in an unsupervised way. This means that the model learns individual facts as probabilistic parameters in contrast to fixed and controlled triples of a KB. Hence, the answer to factual questions may not always be correct and may not be reproducible. Still, the reliability, amount, and quality of generic knowledge "stored" within such models is already remarkable and is improving with its size and training intensity (Roberts et al., 2020). But more importantly, through the training process, an LLM has learned about real-world concepts and their relationships, quite similar to a huge probabilistic ontology. Thus, it is quite capable of understanding natural language, in particular understanding its semantics and coherencies. This important capability is very useful for coping with imprecise user questions enriched with technical context. For instance, future users might not know the exact terms and wording used in original manuals. A traditional KB might need a very extensive ontology to answer such questions meaningfully (and comprehensively), if possible, at all. Furthermore, using a KB requires users to formulate exact queries in a defined (technical) language. In contrast, an LLM processes imprecise natural language requests but also can "translate" its answers in various ways, for instance into actionable commands. These rather generic capabilities have already shown promising results in playing computer games or solving computer tasks (Kim et al., 2023). Given that LLMs are able to act as a kind of KB (even though not always precisely and fully reliable) and most importantly, that LLMs provide a very approachable interface for human users to query this KB, they appear as promising tools to bridge the timespan between today's (and yesterday's) software and computer systems and future users.

Interactive LLM-Assistant Architecture

To assess the feasibility of utilizing LLMs to leverage existing knowledge in a context-aware manner, we have developed a system that integrates LLMs to assist users of an emulated computer environment (see Figure 1) consisting of primary (e.g., the software the user wants to interact with) and secondary (e.g., the operating system) preserved objects. The primary objective is to assist users in managing obsolete software by using natural language as the main interaction mechanism. Users are able to operate directly with the guest system but are also able to delegate a task to the LLM assistant. If a task is submitted, the user's (or more precisely the emulated computer system's) current context is determined and attached to the task.

Additionally, the user's task can be forwarded to a manuals database, containing a collection of documentation material.

LLMs are prone to hallucinations; that is, an LLM will always create an output, even if there is no "knowledge," meaning the LLM had very little or no training data for a given concept. Due to their probabilistic nature, any probability, even a very low one, will lead to a prediction (text output) and thus, it may produce random content in certain situations. In our scenario, however, there is usually plenty of documentation available in the form of unstructured text. Vector database technology can identify relevant text passages based on an input query (in our case, the user query) (Lewis et al., 2020). This text can then be added as additional context. In many standard cases, for example, when working with common operating systems (i.e., secondary objects) such as MS-DOS, OS/2, or older Windows versions, the currently available LLMs possess enough knowledge to provide sufficiently qualified answers. For specialized software products, such as those from research domains (i.e., primary objects), however, there is less knowledge available and thus the results are less accurate. It is also likely that future models might "forget" information about older systems, even the popular ones, since more recent topics will dominate the training sets.

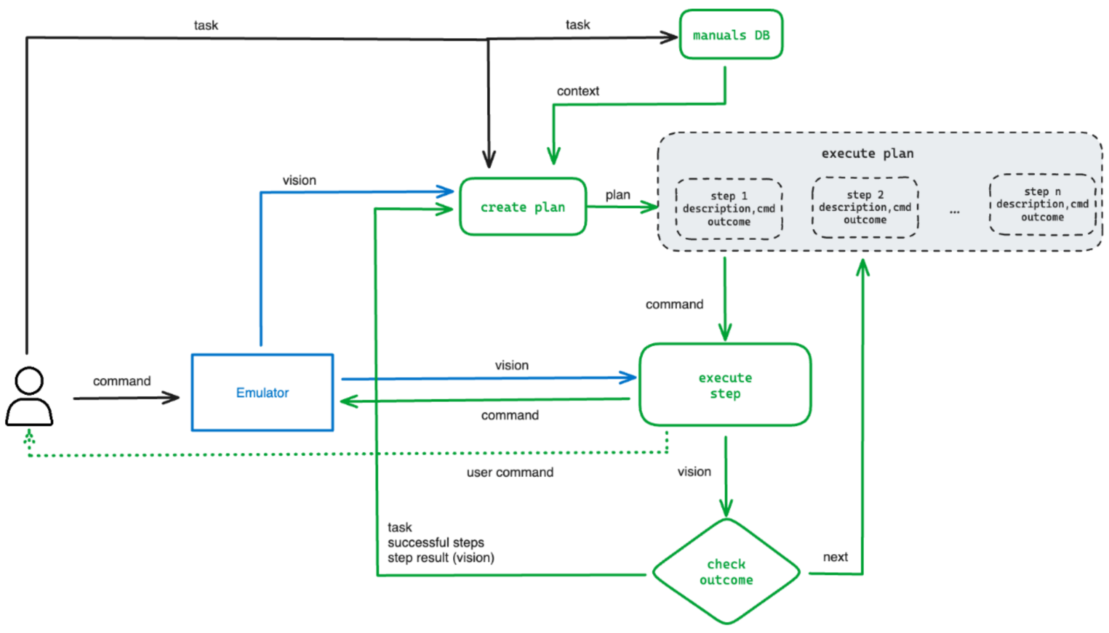


Figure 1: Architecture overview of an LLM assistant. All green components involve LLMs.

Based on the given context and the user's requested task, we prompt the LLM to devise a detailed "plan" comprising steps to execute the user's task. Once a plan is available, the plan is executed step-wise, either fully automated or in cooperation with the user. We have built an experimental setup that connects the Python-based LLM assistant to a QEMU emulator running via the Virtual Network Computing (VNC) protocol[4]. The VNC protocol allows us to retrieve the screen output and to send input events (keyboard input and mouse movements) to the computer system running in the emulator.

While LLMs and associated technologies have significantly advanced, implementing a fully automated version of the proposed system remains challenging. Our initial prototype aims to conduct a technical feasibility study, requiring a simplified experimental setup to perform experiments effectively. These simplifications should not be seen as limitations to the general applicability of LLMs for this purpose. Instead, they highlight specific, current (temporary) technical obstacles that can be overcome without compromising the fundamental concept. This

[4] More formally called the Remote Framebuffer (RFB) protocol.

approach enables us to focus on refining and validating the system's core functionalities while paving the way for addressing more complex challenges in future developments.

Vision – Describing the User-Context

Given that the LLMs in our study are generic language models, the primary task of this component is to convert the computer system's (or software's) appearance or output into a textual representation, serving as contextual information for the LLM. Our focus on language models aims not only at simplifying our experimental setup but more importantly is due to the fact that much of the past and present computer documentation is text-based. In certain domains, such as gaming, this is shifting or has already shifted, with a pronounced trend towards videos and screencasts. In the future, this material will serve as a valuable resource for training image-based neural network models, enhancing the understanding of graphical contexts.

We use two different methods to create a text-based "vision" context: OCR (Optical Character Recognition) and a multimodal LLM (Radford et al., 2021) to describe the content on the screen. For OCR, we have used Tesseract[5] with default settings and PaddleOCR[6]. Both OCR libraries have their strengths: Tesseract was reliable and efficient in extracting longer text passages, while PaddleOCR was better at identifying labels and describing the screen's layout. The multimodal LLMs were GPT-4 Vision[7] and LLaVA[8]. With the exception of GPT-4 Vision, all vision components have been executed on a local machine. Both (conventional) OCR systems or using multimodal LLMs as a (computer) vision component have advantages and shortcomings. The ability to translate an image into a description through configurable prompts is useful for input context and acts as a starting point for planning tasks. It is also beneficial as a description of the output to be compared with the expected result. However, even state-of-the-art models are currently unable to locate specific user interface (UI) elements, e.g., the pixel-exact location of the "next" button. To locate labeled UI elements and for pure text-based systems, we use OCR to convert screen output into a format that the LLM can interpret. We currently employ two distinct outputs from the OCR system: the plain text of the screen, which contains no layout information and is useful for context enrichment; and the layout information, which contains individual text elements with their coordinates described as a bounding box. The latter is particularly useful for extracting target coordinates for mouse clicks.

Even though both methods are very simple, our main objective was to use generic methods suitable for systems from different technical eras and different languages. For text-based applications and graphical user interfaces using labeled UI elements (i.e., typical desktop use cases), these methods provide accurate description and location information to assist the user. For more demanding use cases, such as playing computer games, the processing of the screen is too slow. For UIs with symbolic instead of text representations, these methods do not yield actionable context.

AI Assistance & Planning

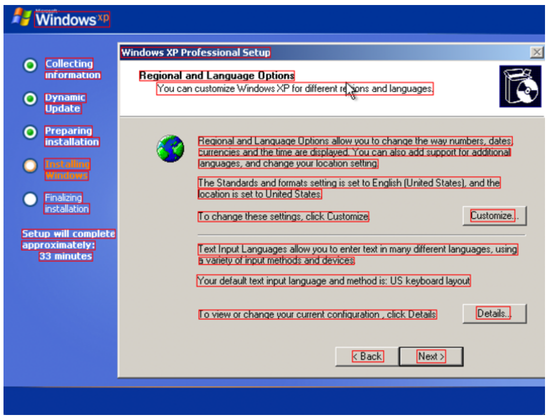
Users can leverage the capabilities of Large Language Models (LLMs) in one of two ways: they can either request the LLM to explain the context and suggest potential actions, or they can direct the LLM to execute a specific task, such as launching a program, opening a file, or installing and configuring software. Additionally, the LLM is capable not only of suggesting input actions but also of predicting and articulating the resulting system output; that is, what will or should appear on the screen. In both scenarios, we utilize the LLM's ability to transform information it was trained on, along with additional contextual information, into an actionable result. For a given task, we instruct the LLM to output a plan with individual steps. Each step

[5] Tesseract, https://github.com/tesseract-ocr/tesseract
[6] PaddleOCR, https://github.com/PaddlePaddle/PaddleOCR
[7] GPT4 Vision, https://openai.com/research/gpt-4v-system-card
[8] LLaVA, https://llava-vl.github.io/

includes a command or interaction to be performed, a description explaining the rationale behind the command, and finally, a description of the expected outcome.

Example: A single step of a Windows XP installation task is presented. The OCR layout information is visualized in the screenshot on the left (Figure 2). All recognized labels are highlighted with red bounding boxes and are provided as JSON text to the LLM, the particular part containing the "Next >" label looks like {*'bounding_box': [[966, 802], [1036, 805], [1034, 833], [965, 829]], 'text': 'Next >'*}. The following additional context provided to the planning step was created by GPT-4 Vision:

> You are viewing a screenshot from the Windows XP Professional Setup process, specifically at the "Regional and Language Options" step. This screen allows you to customize settings related to languages, formats, locations, and input methods.
>
> Here's a breakdown of the information and actions available to the user:
> 1. Regional and Language Options Description: This text explains what the user can do on this screen, which includes changing how numbers, dates, currencies, and time are displayed, as well as adding support for additional languages and changing the location setting.
> 2. Current Settings Summary: It indicates that the current standards and formats are set to English (United States), with the location set to the United States.
> 3. Customize Button: Clicking the "Customize..." button will allow the user to modify regional settings, including formats for numbers, currencies, dates, and times.
> 4. Text Input Languages: This refers to the ability to enter text in many different languages using various input methods and devices, with the default text input language and method being the US keyboard layout.
> 5. Details Button: Clicking the "Details..." button will allow the user to view or change the current text input and keyboard layout configurations.
> 6. Progress Bar: On the left side of the screen, a vertical progress bar indicates the current status of the setup process. It shows that information has been collected, a dynamic update has been completed, and that the system is now in the "Installing Windows" phase.
> 7. Setup Completion Time: Below the progress bar, there is an estimated time for the setup completion, which is approximately 33 minutes.
> 8. Navigation Buttons: At the bottom of the window, there are two navigation buttons "< Back" which allows the user to return to the previous step in the setup process, and "Next >" which would advance the user to the following step after making any desired changes on the current screen.
>
> The user is expected to review and modify these settings if necessary before proceeding with the installation. Once satisfied with the configuration, clicking "Next" would commit the changes and continue with the setup process.


Windows XP Professional Setup

Regional and Language Options

You can customize Windows XP for different regions and languages.

Regional and Language Options allow you to change the way numbers, dates, currencies and the time are displayed. You can also add support for additional languages, and change your location setting.

The Standards and formats setting is set to English (United States), and the location is set to United States.

To change these settings, click Customize.

Text Input Languages allow you to enter text in many different languages, using a variety of input methods and devices.

Your default text input language and method is: US keyboard layout

To view or change your current configuration, click Details.

Setup will complete approximately: 33 minutes

```
{
  "command": "mouse",
  "argument": "Next >",
  "description": "Click on the 'Next' button to
proceed with the installation",
    "output": "The system proceeds to the next
installation step"
}
```

Figure 2. A screenshot of a Windows XP installation process with the visualized OCR result (left) and an actionable step of the "install windows XP" plan created by the LLM.

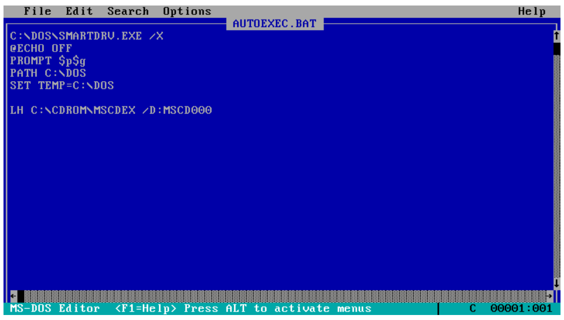
We have instructed the LLM to restrict commands to be produced to precisely three types: keyboard input for character typing; a mouse click on a label identified in the OCR layout information; and, as a fallback, the "user" command. The "user" command enables the LLM to guide the user in executing a specific step when the action is apparent but constraints arise due either to an imperfect vision component or an inability to directly control the emulated computer system's user interface.

Plan Execution and Verification

Each step of the plan can be executed either manually or, in our case, by the VNC subsystem. For this, the JSON keys "command" and "argument" (see Figure 2) are translated into appropriate VNC commands and sent to the guest. Every action, whether initiated by a human user or the VNC subsystem, yields an outcome and may or may not be successful, i.e., yield the expected outcome. Hence, the outcome has to be reviewed, again either by a human user or in an automated way. We have implemented a two-step approach: first, we extract a description of the current machine state and ask an LLM to judge if the description matches the expected outcome of the just-executed step. Here again, we utilize the language flexibility of modern LLMs since both descriptions will never be exactly the same. Since the LLM is able to compare text in an abstract way, it can distinguish very well between an unsuccessful and a successful step. We instruct the LLM to answer with "true" or "false" only, with a default of "false" if a decision could not be made. Even though we saw encouraging results, a thorough evaluation is needed to judge the reliability of this automated assessment. However, in many cases, a misjudgement would not even matter since any subsequent step would fail and fallback to error management would be necessary anyway.

Example: Given the task of adding a greeting message to the autoexec.bat file, the LLM provided detailed five-step instructions. It recommended using the "edit" command and adding the line "echo Hello, welcome!" at the end of the file. Subsequently, we manually executed the suggested steps and asked the model to evaluate the success of this action. The input consisted of the original instruction, the anticipated output, and the OCR result. As can be seen in various examples, OCR output often contains errors. Nevertheless, the LLM managed to parse and interpret even the truncated output from the MS-DOS editor accurately. Given that our agent possessed additional contextual information indicating that this output was a direct consequence of executing "edit autoexec.bat", it accurately and consistently identified successful input. In cases with less output context, however, its success rate was lower.

However, due to shortcomings in vision, the LLM is unable to determine how to reach the "end of the file." Consequently, automated editing will fail and must be performed by the users, as described above. After successfully navigating to the end of the file and inserting the suggested string, control is handed back to the LLM for plan execution, specifically to save the file and close the editing program.



```
C:\DOS\SMARTDRV.EXE /X
@ECHO OFF
PROMPT $p$g
PATH C:\DOS
SET TEMP=C:\DOS

LH C:\CDROM\MSCDEX /D:MSCD000
```

```
MS-DOS Editor  <F1=Help> Press ALT to activate menus    C  00001:001
```

```
File Edit Search Options
AUTOEXEC . BAT
C:\DOS\SMARTDRU.EXE 7X
@ECHO OFF
PROMPT Sp$g
PATH C:N\DOS
SET TEMP=C :\DOS

LH C:\CDROM\MSCDEX 7D: MSCDOOO
```

Figure 3. The screenshot on the left shows the MS-DOS *edit* tool with the open file autoexec.bat. On the right the output of the OCR analysis is shown.

In instances where there is a negative judgment or any other error during the execution of a step, the user is prompted to make a decision. The user has the option to determine whether the task was successful despite the automated assessment's negative outcome or can choose to repair or manually redo the current step. Since it is assumed that users might not always possess the necessary knowledge or skills, they have the option to issue a “replan” request. In such instances, the original task is enhanced with previously successful steps and the current context. A particular issue arose when the output was not classified correctly, sometimes resulting in lost context. For example, when the initial task was formulated just after the system startup, the LLM could deduce the DOS version from the first screen. In these scenarios, the suggested commands almost always matched the DOS version correctly. However, in cases of replanning or when tasks were given with very little additional context (sometimes only the DOS prompt), results were less satisfactory.

Preliminary Results and Conclusion

Our preliminary results already demonstrate the potential of using emulation as a tool to access and render digital objects. Large Language Models (LLMs) offer a significant opportunity to offload and preserve operational knowledge. Furthermore, LLMs provide an abstract and timeless user interface for obsolete software. Our initial prototype shows that LLMs can be integrated seamlessly into user sessions with emulators, offering not only assistance but also a user-friendly and timeless interaction model through natural language. Particularly when the user follows the detailed instructions of the LLM, even this basic setup shows promising results for assisting with the operation of obsolete computer systems. As a subsequent step, we instructed the LLM to produce output in a machine-actionable JSON format and, whenever possible, generate actionable commands. When direct actionable commands were not feasible, detailed instructions were provided to the user.

However, there are both technical and conceptual challenges to address. The primary technical constraint is the system's limited visual capabilities. To address this, improved image-to-text models capable of translating visual contexts into textual descriptions are necessary to offer guidance and automated interaction, also for more complex environments such as computer games.

Another challenge lies in the inherent limitations of Large Language Models (LLMs). While they often deliver remarkable results, their output is not always reproducible or consistent, and occasionally it can be nonsensical. Fine-tuning a model specifically for this task is difficult, primarily due to the lack of high-quality training data. Additionally, from a conceptual standpoint, concentrating on a specialized model might be counterproductive. It is uncertain what specific usage challenges will arise in the future and what kind of additional support and knowledge will be necessary. The primary advantage of LLMs in this setting is their broad-based approach combined with their capabilities to manage and digest natural language tasks. This allows for an efficient translation of a user's task, articulated in natural (and possibly non-expert) language, into a detailed, partially actionable technical guide.

Acknowledgements

Part of the activities and insights presented in this paper were made possible through the NFDI consortium DataPLANT, 442077441, supported through the German National Research Data Initiative (NFDI 7/1).

References

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., Nori, H., Palangi, H., Ribeiro, M. T., & Zhang, Y. (2023). *Sparks of Artificial General Intelligence: Early experiments with GPT-4* (arXiv:2303.12712). arXiv. https://doi.org/10.48550/arXiv.2303.12712

Fiore, D., Thiel, C., & Baldauf, M. (2020). Potenziale von Chatbots für den innerbetrieblichen IT-Support. *HMD Praxis der Wirtschaftsinformatik*, *57*(1), 77–88. https://doi.org/10.1365/s40702-019-00578-7

Kim, G., Baldi, P., & McAleer, S. (2023). *Language Models can Solve Computer Tasks* (arXiv:2303.17491). arXiv. https://doi.org/10.48550/arXiv.2303.17491

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., & Rocktäschel, T. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. In H. Larochelle, M.A. Ranzato, R. Hadsell, M.F. Balcan & H.T. Lin (Eds.) Advances in Neural Information *Processing Systems 33: Annual Conference on Neural Information Processing Systems* (pp. 9459-9474). Retrieved from: https://researchr.org/publication/nips-2020.

Loch, F., & Stolze, M. (2023). An Intuitive Interface for Technical Documentation Based on Semantic Knowledge Graphs. In H. P. da Silva & P. Cipresso (Eds.), *Computer-Human Interaction Research and Applications* (pp. 307–317). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-49425-3_19

Oberhauser, J., Gieschke, R., & Rechert, K. (2022). Automation is Documentation: Functional Documentation of Human-Machine Interaction for Future Software Reuse. *International Journal of Digital Curation*, *17*(1). https://doi.org/10.2218/ijdc.v17i1.836

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., & Clark, J. (2021). Learning transferable visual models from natural language supervision. In M. Meila & T. Zhang (Eds.) *International Conference on Machine Learning* (pp.8748-8763). PMLR 139 Retrieved from http://proceedings.mlr.press/v139/, 8748–8763.

Rettig, M. (1991). Nobody reads documentation. *Communications of the ACM*, *34*(7), 19–24. https://doi.org/10.1145/105783.105788

Roberts, A., Raffel, C., & Shazeer, N. (2020). *How Much Knowledge Can You Pack Into the Parameters of a Language Model?* (arXiv:2002.08910). arXiv. https://doi.org/10.48550/arXiv.2002.08910

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., & Lample, G. (2023). *LLaMA: Open and Efficient Foundation Language Models* (arXiv:2302.13971). arXiv. https://doi.org/10.48550/arXiv.2302.13971

Vrandečić, D., & Krötzsch, M. (2014). Wikidata: A free collaborative knowledgebase. *Communications of the ACM*, *57*(10), 78–85. https://doi.org/10.1145/2629489