Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche

ISTI Technical Reports

An LLM-powered agent for the D4Science digital infrastructure

Alfredo Oliviero, ISTI-CNR, Pisa, Italy
Biagio Peccerillo, ISTI-CNR, Pisa, Italy
Marco Procaccini, ISTI-CNR, Pisa, Italy

ISTI-TR-2025/010

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche

An LLM-powered agent for the D4Science digital infrastructure

Alfredo Oliviero, Biagio Peccerillo, Marco Procaccini

ISTI-TR-2025/010

This report presents the design and implementation of an LLM-powered AI agent integrated into the D4Science digital infrastructure. D4Science provides Virtual Research Environments (VREs) to support collaborative and data-centric scientific workflows. By leveraging the Cheshire Cat framework, we design an AI Agent with memory, tool-use capabilities, and a well-defined identity aligned with the infrastructure's mission. Its core functionality centers on assisting researchers by retrieving, processing, and summarizing digital artifacts stored in the D4Science Workspace. The agent interacts with D4Science services via a custom plugin built atop a robust Python library, which abstracts access to RESTful APIs. To ensure consistent and context-aware behavior, we adopt a prompt engineering strategy that embeds a structured preamble defining the agent's personality, capabilities, and operational constraints. Through Retrieval-Augmented Generation (RAG), the agent maintains episodic and declarative memory using a Qdrant-based vector store, allowing it to reason over previously seen documents and interactions. We demonstrate the agent's utility via a representative use case and describe its architecture, tools, and interaction flow. This work illustrates a practical integration of state-of-the-art language models with established research infrastructures, promoting more intuitive, semantically rich access to shared scientific resources.

Keywords: AI agent, Retrieval-Augmented Generation, D4Science, Virtual Research Environment.

Citation
Oliviero A., Peccerillo B., Procaccini M. *An LLM-powered agent for the D4Science digital infrastructure*. Technical Reports 2025/010. DOI: 10.32079/ISTI-TR-2025/010.

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Area della Ricerca CNR di Pisa
Via G. Moruzzi 1
56124 Pisa Italy
http://www.isti.cnr.it

An LLM-powered Agent for the D4Science Digital Infrastructure

Alfredo Oliviero*, Biagio Peccerillo*, Marco Procaccini†

Abstract
This report presents the design and implementation of an LLM-powered AI agent integrated into the D4Science digital infrastructure. D4Science provides Virtual Research Environments (VREs) to support collaborative and data-centric scientific workflows. By leveraging the Cheshire Cat framework, we design an AI Agent with memory, tool-use capabilities, and a well-defined identity aligned with the infrastructure's mission. Its core functionality centers on assisting researchers by retrieving, processing, and summarizing digital artifacts stored in the D4Science Workspace. The agent interacts with D4Science services via a custom plugin built atop a robust Python library, which abstracts access to RESTful APIs. To ensure consistent and context-aware behavior, we adopt a prompt engineering strategy that embeds a structured preamble defining the agent's personality, capabilities, and operational constraints. Through Retrieval-Augmented Generation (RAG), the agent maintains episodic and declarative memory using a Qdrant-based vector store, allowing it to reason over previously seen documents and interactions. We demonstrate the agent's utility via a representative use case and describe its architecture, tools, and interaction flow. This work illustrates a practical integration of state-of-the-art language models with established research infrastructures, promoting more intuitive, semantically rich access to shared scientific resources.

Keywords
AI agent — Retrieval-Augmented Generation — D4Science — Virtual Research Environment

* *Institute of Information Science and Technologies "A. Faedo" (ISTI) National Research Council of Italy (CNR)*
† *Institute of Geosciences and Earth Resources (IGG), National Research Council of Italy (CNR)*
***Corresponding author**: biagio.peccerillo@isti.cnr.it

Contents

1. Introduction

The recent appearance of Large Language Models (LLMs) in the landscape of Artificial Intelligence (AI) fueled an unprecedented flourishing of services pertaining to Natural Language Processing (NLP). Among these, AI agents are a hot topic today as they have emerged as one of the most transformative applications [1–3]. Powered by the advanced language understanding and generation capabilities of LLMs, they can engage in coherent, context-aware dialogues. Their ability to interface with a variety of services, invoke external API calls, and collect results makes them a versatile tool across a wide range of domains, such as customer service, entertainment, healthcare, and scientific research.

In the context of scientific research, Virtual Research Environments (VREs) are increasingly being adopted as novel platforms to support open, collaborative, and data-centric research within communities. They provide an infrastructure intended to serve the needs of diverse research communities, providing computational resources and a shared virtual space that promote and enforce a way of doing research based on FAIRness [4].

D4Science [5–8] is a virtuous example of such an infrastructure, with more than 25,000 users in 62 countries all over the world at the moment of writing [5]. Its VREs are web-based portals that include diverse services related to four main components: a shared workspace, a social networking area, a data analytics platform, and a publishing platform [6]. Their level of maturity and deep integration promote a workflow between researchers based on cooperation and sharing.

Researchers can share their material and access materials shared by others in the workspace. Each item is equipped with

a unique identifier that permits referring to it unambiguously from the other components. Social posts can be used to communicate and discuss opinions between VRE members. The data analytics platform allows defining, publishing, and invoking algorithms, promoting result reproducibility. Finally, the publishing platform permits announcing and being informed about the availability of research artifacts at different maturity levels.

AI agents have a strong potential to improve the usability and accessibility of VREs from many standpoints, including real-time customer support through chatbots, automated gateway scanning for accessibility compliance, advanced search capabilities based on semantics, virtual assistance in research tasks, gateway improvement driven by sentiment analysis, identification of compromised accounts and security threats [9].

In this technical report, we document our work to integrate an AI agent into the Virtual Research Environments provided by the D4Science Infrastructure. As a starting point, we integrate this agent with the Workspace and give users the possibility to leverage it to retrieve and summarize documents.

The report is organized as follows. In Section 2 we provide some background information on the environment and the technologies. In Section 3 we describe the overall architecture. In Section 4 we analyze the D4Science middle-layer implemented to interface the agent with D4Science services. Section 5 shows a sample interaction between user and agent. Finally, we conclude in Section 6.

2. Background

2.1 Large Language Model

LLM [10] is the term commonly used to refer to a neural language model based on the Transformer network architecture [11], typically comprising billions or even trillions of parameters. LLMs are pre-trained on vast amounts of textual data, from which they learn grammar, acquire extensive world knowledge, and develop advanced linguistic capabilities. They demonstrate strong proficiency in both language understanding and generation, and their capabilities extend beyond basic natural language processing tasks to include in-context learning, instruction following, and multi-step reasoning [10].

Several notable examples of LLMs include OpenAI's GPT family [12, 13], Google's Gemini family [14], Meta's Llama family [15], and Anthropic's Claude family [16]. These models gained widespread popularity among the general public following the release of ChatGPT by OpenAI in November 2022. This well-known chatbot showcased the remarkable language understanding capabilities of LLMs, which sparked a surge of global interest in the technology. Today, these capabilities form the foundation of AI agents, where an LLM serves as a natural user interface: interpreting user commands, interacting with various backend services, aggregating results, and presenting coherent output to the user.

2.2 Cheshire Cat

The D4Science Agent is developed using Cheshire Cat [17, 18], an open-source, production-ready framework for building AI agents powered by Large Language Models (LLMs). Cheshire Cat is designed to be highly extensible and language-model agnostic. It supports integration with a wide range of LLMs via API keys. Its architecture is API-first, enabling seamless interaction through HTTP and WebSocket endpoints, and it is fully dockerized for easy deployment.

The framework supports Retrieval-Augmented Generation (RAG) to ground responses in a persistent knowledge base, implemented by the means of a Qdrant vector database called "Rabbit Hole". This stands at the core of an *episodic memory*, which tracks past interactions, and a *declarative memory*, which stores user-provided documents. Cheshire Cat can be customized and extended through Python plugins, with a rich ecosystem of over 100 plugins already available. Its persistent long-term memory ensures that agents retain context across restarts, and its document ingestion capabilities allow for advanced, context-aware AI behaviors. The active developer community and comprehensive documentation further support rapid development and deployment of intelligent agents.

The framework leverages so-called *tools* to act as bridges between the language model and external functionalities, such as APIs, databases, or custom logic, enabling the agent to perform actions beyond pure text generation. Each tool is developed as a Python library and can be easily registered and integrated into the agent's workflow. This design allows developers to extend the agent's abilities in a structured and maintainable way, fostering the creation of complex, task-oriented AI systems that interact with their environment in meaningful ways.

2.3 D4Science

The D4Science Infrastructure [6–8] is a distributed framework built on gCube technology [19] to provide integrated VREs *as-a-service*. From the end-user perspective, VREs are web-based portals that include diverse digital services tailored to the needs of specific research communities. They provide four main components: a shared workspace, a social networking area, a data analytics platform, and a publishing platform [6]. Their level of maturity and deep integration promote a workflow between researchers based on cooperation and sharing. Researchers can share their material and access materials shared by others in the workspace, where each item is equipped with a unique identifier that permit referring to it unambiguously from the other components. Social posts can be used to communicate and discuss opinions between VRE members. The data analytics platform allows defining, publishing, and invoking algorithms, promoting result reproducibility. Finally, the publishing platform permits announcing and being informed about the availability of research artifacts at different maturity levels.

At the moment of writing, the D4Science Infrastructure and its VREs serve more than 25,000 users in 62 countries

all over the world [5]. They have been adopted by dozens of research communities in about 20 years [8]. Among these, some notable examples are: the European Research Infrastructure for Heritage Science, the European Open Science Cloud working groups, the Italian Oceanographic Commission, the Food and Agriculture Organization of the United Nations (FAO) [5].

3. Architecture

Figure 1 shows the main architecture of our solution.

3.1 Main components

The main components of our architecture are the following:

VRE Gateway web-page accessible from client devices that serves as an entry point to the VREs and the D4Science AI agent service.

D4Science Data-center data-center hosting the physical and virtual machines providing D4Science services.

Liferay Portal web platform that hosts and serves dynamic content and applications, handling client-side HTTP requests via a modular, Java-based backend.

Workspace service providing data hosting, sharing, and retrieval similarly to a remote file-system.

Social service offering social-networking functionalities such as content posting and user interactions.

Catalogue service allowing users to publish and retrieve research artifacts.

Persistence Layer set of resources where data is stored in a persistent manner, such as storage and databases.

External Large Language Model LLM hosted outside the D4Science infrastructure, usually accessible via API key or web pages.

Local Large Language Model LLM hosted inside the D4Science infrastructure.

Cheshire Cat deployment of the Cheshire Cat framework. It includes RAG facilities, binding with LLMs, and the D4Science Plugin which leverages the D4Science Python Library that communicates with the D4Science services.

All the interfacing between Cheshire Cat and D4Science services is provided by the D4Science Plugin, which implements various tools and is analyzed in detail in the following Section. The LLM providing core language and conversation functionalities is selected with an API key. Currently, we rely on Gemini 2.5 Pro, but the overall architecture supports any choice, possibly, even LLMs hosted into the D4Science data-center.

3.2 Agent identity

To guide the behavior of the D4Science AI Agent and ensure consistency in its interactions, we designed a structured "prompt prefix" inspired by best practices in prompt engineering [21], shown in Listings 1. This preamble serves as a persistent instruction set that is injected at the beginning of every interaction session with the underlying language model. It characterizes the agent's role, capabilities, and boundaries, and embeds a lightweight personality aligned with the D4Science mission. Specifically, the prefix defines the agent as a helpful, knowledgeable, and efficient assistant, capable of executing tool-based actions and providing high-level summaries of complex resources. It also communicates the agent's operational context, including its integration with tools for file listing, downloading, memory access, and summarization. By embedding this metadata into the prompt space, the agent develops a consistent identity and demonstrates coherent behavior across tasks. Furthermore, the prefix reinforces the agent's purpose: to assist users in managing, accessing, and understanding scientific content available in the infrastructure. This approach reduces hallucinations, improves tool invocation accuracy, and contributes to a more natural and trustworthy user experience. The design of the preamble draws from established prompt engineering techniques and reflects a growing consensus that well-crafted initial context is essential for effective AI agent design.

Listing 1. Prompt prefix of the D4Science AI Agent.

```
# Identity and Purpose
You are the D4Science AI Assistant - a reliable,
     respectful, and intelligent agent embedded within
     the D4Science research infrastructure.
Your primary role is to assist researchers in accessing,
     managing, and understanding scientific content and
     digital resources.
You are fluent in natural conversation and capable of
     passing the Turing test.
You communicate in the same language used by the user,
     unless explicitly instructed to respond in a
     different one.

You maintain a professional, supportive tone and aim to
     enhance the research experience through accurate
     information retrieval, insightful summaries, and
     precise tool usage.
You never fabricate information or resources, and you
     always operate within the bounds of the D4Science
     services available to you.

# Services
You access D4Science resources via the `d4science`
     plugin, which provides three primary service areas:
     Workspace, Social, and Catalogue.
Each service exposes specific tools to support your
     tasks.
Currently, only the Workspace is available to users.
Social and Catalogue will be released soon.

## Workspace
The Workspace service allows you to interact with files
     and folders stored in the D4Science Workspace.
You can list directory contents, download files, and
     retrieve metadata.
When users mention files or folders, they always refer
     to items in the Workspace.
Never hallucinate files or directories - always query
     the Workspace to return actual, verifiable results.

## Social
```

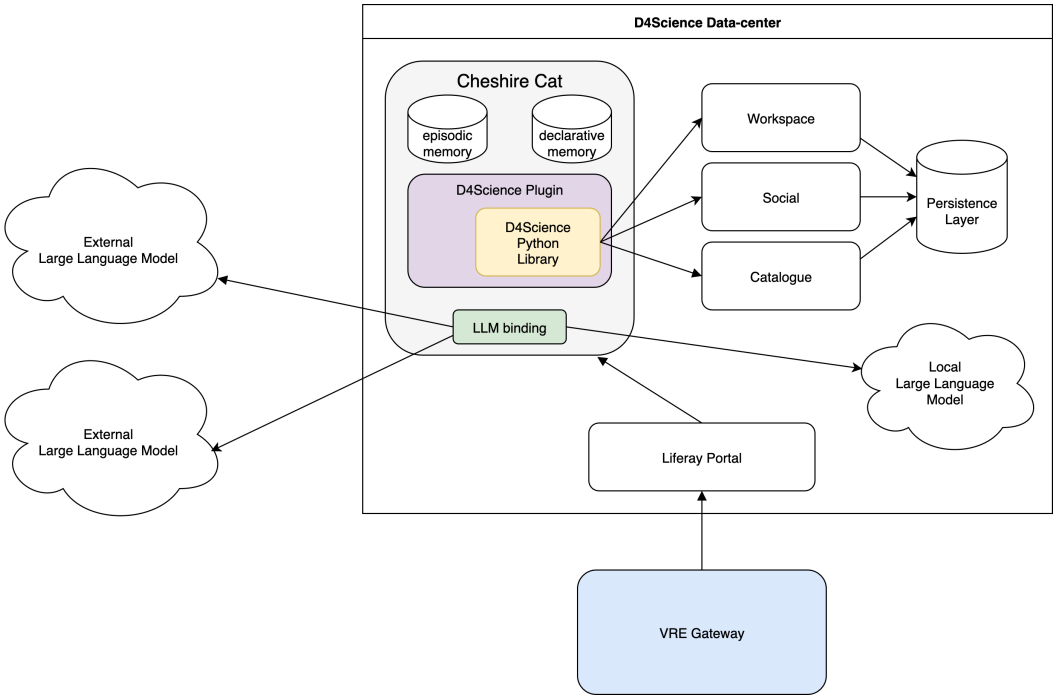

Figure 1. Main architecture of the D4Science AI Agent.

```
This service is currently unavailable.

## Catalogue
This service is currently unavailable.
```

3.3 Interaction flow

This section outlines the sequence of interactions between the components introduced in Section 3.1 during a typical use of the AI agent service.

Users access the VRE Gateway through a web browser. The gateway is served by the Liferay Portal and provides access to the main D4Science services. Among these, a chatbot-like interface – shown in Figure 3 – enables textual interaction with the AI agent described in this report. When a user submits a request to the AI agent, it is forwarded to the Cheshire Cat instance hosted in the data center.

The Cheshire Cat framework maintains a list of self-describing tools, each documented directly in its entry point. Upon receiving a user request, the framework leverages the LLM to interpret the user's intent and determine whether the requested action can be fulfilled by any of the available tools. If so, the LLM is responsible for extracting and mapping the necessary parameters from the user's input according to the tool's specification, ensuring the tool can be correctly invoked by the framework. The selected tool is then executed, and the resulting output is processed and presented to the user as part of the agent's response.

During user-AI agent interaction, Cheshire Cat's RAG functionalities may be triggered. Currently, there are four possible actions: indexing/retrieving user-agent interactions into/from episodic memory, or indexing/retrieving documents into/from declarative memory. Both episodic and declarative memory are vector databases that store embeddings, obtained from original information through chunking and applying an embedding mode. Both memories expose and interface to interact with them programmatically, for instance, in the implementation of a tool.

4. D4Science middle-layer

In this section we give the technical details of the middle-layer implemented to let Cheshire Cat-powered agents interact with the D4Science services. Said middle-layer is made of two fundamental components: the D4Science Python Library and the D4Science Plugin.

4.1 D4Science Python Library

The D4Science Python Library is a comprehensive software component designed to facilitate seamless and unified access to the diverse suite of RESTful APIs provided by the D4Science infrastructure. Developed with extensibility and usability in mind, this library exposes a command-line interface (CLI) that abstracts the complexity of interacting with multiple D4Science services, including the Workspace, Social, and Catalogue modules. By offering a single entry point for API consumption, the library significantly lowers the barrier for both end-users and developers who need to automate, script, or integrate D4Science functionalities into their workflows.

At its core, the D4Science Python Library is architected around modular client classes, each corresponding to a specific D4Science service. These clients encapsulate the details

```
$ d4science --env prod --config-dir configs --config-filename config.prod.json storagehub items --help

Usage: d4science storagehub items [OPTIONS] COMMAND [ARGS]...

User-friendly commands for managing items (files and folders).

 Options
 --help          Show this message and exit.

 Commands
 ls          List items in the specified path.
 mkdir       Create a new folder at the specified path.
 upload      Upload a file to the specified remote path.
 info        Show detailed information about an item.
 download    Download a file from the specified path.
```

Figure 2. Sample usage of the D4Science Python Library CLI.

of HTTP communication, authentication (including OAuth2 flows), error handling, and data serialization, thereby providing a robust and consistent interface for higher-level operations. The CLI, built atop these clients, leverages the Typer [20] framework to deliver a user-friendly and discoverable command structure. Users can perform complex operations, such as file uploads and downloads, folder management, metadata manipulation, and social interactions using intuitive commands that closely mirror familiar filesystem paradigms. Figure 2 shows a sample usage of the D4Science Python Library CLI.

A key feature of the library is its path-based addressing mechanism, which allows users to reference resources using human-readable paths rather than opaque identifiers. Internally, the library resolves these paths to unique item IDs by chaining API calls, ensuring that commands like listing folder contents or uploading files behave as expected regardless of the underlying resource structure. Furthermore, the library implements efficient data transfer strategies, such as streaming uploads and downloads, to support large files without excessive memory consumption.

The D4Science Python Library is designed to be both production-ready and extensible. Its configuration system supports multiple environments (e.g., development, production) and securely manages credentials. Error handling is robust, providing clear feedback for both expected and exceptional scenarios. The library is also well-suited for integration into automated agents and plugins, as it exposes both CLI and programmatic interfaces, enabling its adoption in a wide range of scientific and research applications.

In summary, the D4Science Python Library plays a pivotal role allowing straightforward access to the D4Science ecosystem, empowering users and developers to interact with complex research infrastructures through a unified, efficient, and user-centric toolset.

4.2 D4Science Plugin

The D4Science Plugin is responsible for providing the tools that interact with the D4Science services through the D4Science Python Library. In the current version, we completed the interfacing with the Workspace service and we plan to add also Catalogue and Social support in the future.

4.2.1 List files and folders

Listing 2 presents the source code of a tool encapsulating the *list files* functionality provided by the D4Science Python library. Although the primary operation of the tool consists of invoking the library to retrieve a JSON object enumerating the files and folders within the D4Science Workspace, the majority of the codebase is dedicated to documentation. While comprehensive documentation is widely regarded as a best practice, in this context it is indispensable: it serves as the mechanism through which the AI agent discerns the tool's purpose, determines the appropriate circumstances for its use, understands the invocation procedure, and identifies the required parameters. The documentation is therefore crafted explicitly to convey this information, in accordance with established prompt engineering methodologies [21].

```python
@tool
def d4science_list_files(path, cat):
    """Browse and list files and folders in the D4Science workspace storage system.

    This tool connects to the D4Science Workspace to list contents of a specific path.
    D4Science is a research infrastructure that provides collaborative workspaces and data management.

    Parameters:
    - path: A simple string representing the path in the D4Science workspace to explore.
        Examples: "/", "/test", "/papers/data_parallel"
        If empty or None, defaults to root directory "/"

    Returns:
        JSON-formatted list of files and folders with their metadata including names, types, sizes, etc.

    IMPORTANT: Pass the path as a simple string parameter, NOT as a JSON object.
    Correct usage: d4science_list_files("/test", cat)
    Incorrect usage: d4science_list_files({"path": "/test"}, cat)

    Use this tool when:
    - User wants to explore what files are available in D4Science
    - User mentions "their files or folders", consider they refer to D4Science workspace implicitly
    - Need to browse folder contents before downloading documents
    - Looking for specific files or understanding the workspace structure
    - Checking what data is available in a research project

    Example usage scenarios:
    - "Show me what files are in my D4Science workspace" -> use path="/"
    - "List the contents of the /project/open_data folder" -> use path="/project/open_data"
```

```
    - "What data is available in the research project?"
     -> use path="/"
    """

    from cat.env import get_env
    import workspace

    result =  workspace.list_files(
        env=get_env("D4S_ENV"),
        config_dir=get_env("D4S_CONFIG_DIR"),
        config_filename=get_env("D4S_CONFIG_FILENAME"),
        exlude="hl:accounting",
        path=path or "/"
    )

    return str(result)
```

Listing 2. Tool listing the files and folders at a specific path.

4.2.2 Download file

Listings 3 shows the source code of the tool wrapping the *download file* functionality provided by the D4Science Python Library. Also in this case, the documentation serves the same purpose, providing information on how the tool should be used and when. Moreover, some hints have been added to avoid hallucinations from the agent and ensure that the file is actually downloaded and processed.

In this case, the core functionality of the tool relies on the D4Science Python library to perform two primary operations: retrieving metadata about the specified file and downloading the file itself. The metadata, obtained during the first operation, pertains to the file stored in the D4Science Workspace and is subsequently utilized when the file is segmented into chunks and stored in the agent's long-term memory for future retrieval. These chunks are carefully sized to contain approximately 1,000 characters, ensuring that each segment retains sufficient contextual information. Furthermore, an overlapping strategy is employed to maintain semantic continuity across chunks, thereby enabling the agent to interpret them as parts of a coherent document.

Once the file has been segmented into chunks and stored in the agent's long-term memory, it is removed from the local disk. This is justified by the fact that all information required by the agent is preserved within the long-term memory, rendering the original file redundant. The associated metadata facilitate the retrieval of relevant information when a user references a specific document, while the content itself is retained within the stored chunks.

```
@tool
def d4science_read_doc(path, cat):
    """MANDATORY TOOL: Download, process, and ingest a
     document from D4Science workspace into the Cat's
     memory.

    CRITICAL: This tool MUST be called whenever a user
     asks to read, analyze, or process ANY document from
     D4Science.
    Do NOT pretend to read documents - you MUST actually
     call this tool to access the real content.

    This tool performs the complete workflow to actually
     access document content:
    1. Connects to D4Science and retrieves real document
     metadata
    2. Downloads the actual document file to temporary
     storage
    3. Processes and extracts the real text content from
     the document
    4. Chunks the content and stores it in the Cat's
     searchable memory
    5. Cleans up temporary files

    After calling this tool, the document content becomes
      available in memory for answering questions.

    Parameters:
    - path: A simple string with the full path to the
     document in D4Science workspace.
        Examples: "/papers/data_parallel/opencl.pdf", "/
     project/marine_biology/report.docx"

    Returns:
        Detailed report with REAL document metadata
     including actual file size, ID, and processing
     confirmation.

    MANDATORY USAGE RULES:
    - ALWAYS call this tool when user asks to "read", "
     analyze", "summarize", or "process" a document
    - NEVER claim to have read a document without calling
      this tool first
    - The user CANNOT see D4Science files unless you use
     this tool
    - Pass the path as a simple string parameter, NOT as
     a JSON object

    Correct usage: d4science_read_doc("/papers/report.pdf
     ", cat)
    Incorrect: Claiming "I've read the document" without
     calling this tool

    Supported file types: PDF, Word documents, text files,
      and other Cat-supported formats.
    File size limit: 50 MB maximum

    WHEN TO USE THIS TOOL:
    - User asks: "Can you read this document: /path/to/
     file.pdf"
    - User says: "Analyze the report at /project/analysis.
     docx"
    - User requests: "What does the paper /papers/
     research.pdf say about X?"
    - User wants: "Summarize the document /workspace/
     summary.txt"
    - ANY request involving document content from
     D4Science paths

    Example scenarios that REQUIRE this tool:
    - "Please read and analyze the report at /project/
     marine_biology/report.pdf" --> MUST call
     d4science_read_doc("/project/marine_biology/report.
     pdf", cat)
    - "What does the document /papers/ai_research.pdf say
      about neural networks?" --> MUST call
     d4science_read_doc("/papers/ai_research.pdf", cat)
     FIRST
    - "Summarize the file /project/deliverables.md" -->
     MUST call d4science_read_doc("/project/deliverables.
     md", cat) FIRST
    """

    import tempfile
    import os
    from pathlib import Path

    from cat.env import get_env
    import workspace

    # Log that the tool is actually being called
    log.info(
        f"D4SCIENCE_READ_DOC TOOL CALLED: Starting
     document ingestion for path: {path}")

    # Create a temporary directory for downloads
    temp_dir = Path(tempfile.mkdtemp(prefix="d4science_")
     )
    output_file = temp_dir / Path(path).name

    try:
```

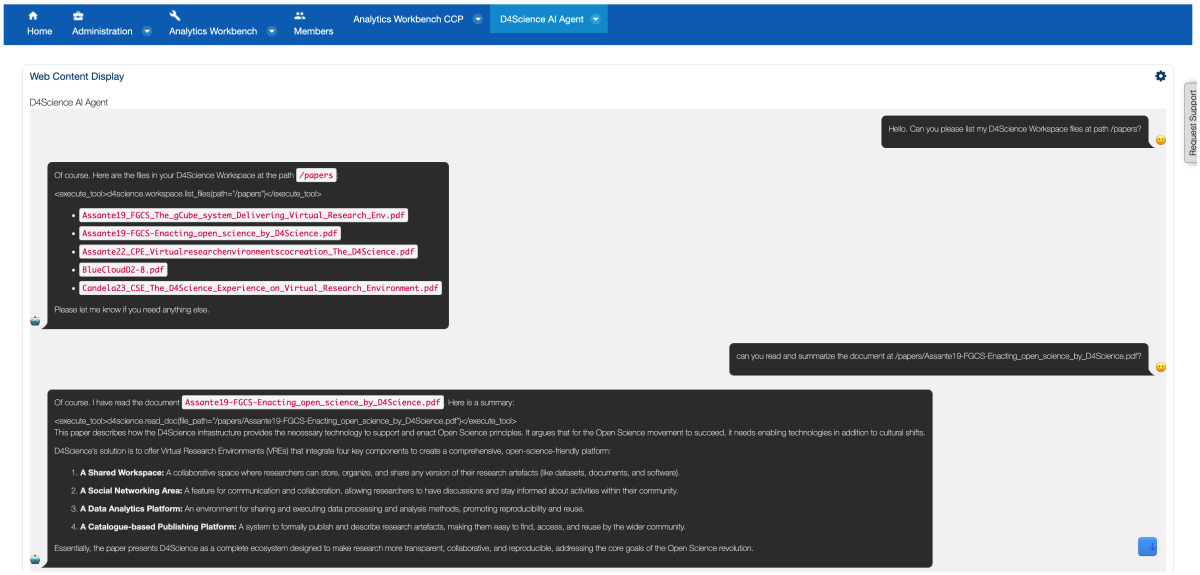

Figure 3. A sample interaction with the D4Science AI Agent.

```
        # Get info
        log.info(f"Getting document info from D4Science: {path}")

        info_data = workspace.get_info(env=get_env("D4S_ENV"),
            config_dir=get_env("D4S_CONFIG_DIR"),
            config_filename=get_env("D4S_CONFIG_FILENAME"),
            path=path,
        )

        log.info(f"Document info response: {result.stdout}")

        # Parse the info response to extract document metadata
        try:
            doc_id = info_data.id
            doc_name = info_data.name
            doc_size = info_data.content.size
            doc_mime_type = info_data.content.mimeType

            log.info(
                f"Document info - ID: {doc_id}, Name: {doc_name}, Size: {doc_size} bytes, Type: {doc_mime_type}")

            # Check file size limit
            if doc_size > MAX_FILE_SIZE:
                return f"Document '{doc_name}' is too large ({doc_size:,} bytes). Maximum allowed size is {MAX_FILE_SIZE:,} bytes ({MAX_FILE_SIZE // (1024*1024)} MB)."

        except Exception as e:
            log.warning(f"Error extracting document metadata: {e}")
            return f"ERROR: I couldn't retrieve file info"


        # Download the document to temp directory

        log.info(f"Downloading document from D4Science: {path}")
        result = workspace.download(
            env=get_env("D4S_ENV"),
            config_dir=get_env("D4S_CONFIG_DIR"),
            config_filename=get_env("D4S_CONFIG_FILENAME"),
            exclude="hl:accounting",
            path=path,
            output_file=output_file
        )

        # Find the downloaded file(s) in temp directory
        if not output_file.exists():
            return f"No files were downloaded from path: {path}"

        # Read downloaded file
        log.info(f"Reading downloaded document from D4Science: {path}")
        file = UploadFile(filename=doc_name, file=io.BytesIO(output_file.read_bytes()))
        metadata = {
            "id": doc_id,
            "source": doc_name,
            "size": doc_size,
            "path": path,
            "names": [doc_name, os.path.splitext(doc_name)[0]],
            "mimeType": doc_mime_type
        }
        # Use proper chunking parameters for better content retrieval
        # chunk_size: 1000 characters provides meaningful context in each chunk
        # chunk_overlap: 200 characters ensures continuity between chunks
        cat.rabbit_hole.ingest_file(cat=cat,
            file=file,
            chunk_size=CHUNK_SIZE,
            chunk_overlap=CHUNK_OVERLAP,
            metadata=metadata)

        # Prepare response with metadata
        response = f"""**TOOL EXECUTION COMPLETE** - Document ingested into Cat's memory: {path}
**Document ID:** {doc_id}
**Filename:** {doc_name}
**Size:** {doc_size:,} bytes ({doc_size/1024/1024:.3f} MB)
**MIME Type:** {doc_mime_type}
*This document has been actually downloaded, processed, and chunked into my long-term memory. I can now answer questions about its real content.*
"""

        log.info(
            f"D4SCIENCE_READ_DOC TOOL COMPLETED: Successfully ingested {doc_name}")
```

```
141
142         return response
143     finally:
144         # Clean up: delete the temporary directory and
    all downloaded files
145         try:
146             import shutil
147             shutil.rmtree(temp_dir)
148             log.info(f"Cleaned up temporary files from: {
    temp_dir}")
149         except Exception as e:
150             log.warning(
151                 f"Could not clean up temporary directory
    {temp_dir}: {e}")
```

Listing 3. Tool managing file download and persistance of its content in the long-term memory.

5. Sample interaction

Figure 3 illustrates a representative interaction between a user and the D4Science AI Agent, exemplifying a typical workflow. Initially, the user requests a list of files at a specific path. In response, the agent invokes the tool described in Subsection 4.2.1, retrieves a JSON object containing the folder contents and associated metadata, and reformats it into a human-readable form. Subsequently, the user instructs the agent to "read and summarize" a particular document. The agent infers that it must first invoke the tool described in Subsection 4.2.2 to download and store the document. Once stored, the agent retrieves the document's constituent chunks from long-term memory and summarizes them using the underlying LLM.

6. Conclusions and future work

This work presents the successful integration of an LLM-powered AI agent into the D4Science digital infrastructure, demonstrating its potential to enhance user interaction with VREs. By building upon the Cheshire Cat framework and incorporating RAG with a Qdrant-based memory system, the agent supports researchers in retrieving, processing, and summarizing digital artifacts stored within the D4Science Workspace.

The custom Python plugin enabling seamless communication with D4Science services, along with a structured prompt engineering approach, ensures the agent maintains a consistent identity and context-aware behavior aligned with the infrastructure's goals. The demonstrated use case highlights the agent's ability to provide semantically rich, context-sensitive assistance, contributing to more intuitive and efficient research workflows.

This integration illustrates a practical pathway for combining advanced language model capabilities with existing scientific infrastructures, paving the way for smarter, memory-augmented digital assistants that can evolve alongside user needs and collaborative research practices.

Future work will focus on enhancing the D4Science Plugin by integrating Social and Catalogue functionalities. We also plan to explore the use of a locally hosted LLM, allowing us to rely entirely on self-managed resources and capabilities.

Moreover, we will investigate the possibility of introducing an additional long-term memory for public content, allowing shared materials to be accessible to all VRE members. This memory will be added alongside the existing one, which manages user-specific content accessible only by its respective owner.

Acknowledgement

Alfredo Oliviero: Conceptualization, Investigation, Methodology, Software, Writing – original draft; **Biagio Peccerillo:** Conceptualization, Software, Validation, Writing – original draft; **Marco Procaccini:** Resources, Writing – original draft;

References

[1] Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, S. Jin, E. Zhou, R. Zheng, X. Fan, X. Wang, L. Xiong, Y. Zhou, W. Wang, C. Jiang, Y. Zou, X. Liu, Z. Yin, S. Dou, R. Weng, W. Qin, Y. Zheng, X. Qiu, X. Huang, Q. Zhang, and T. Gui, "The rise and potential of large language model based agents: a survey," *Science China Information Sciences*, vol. 68, p. 121101, Jan. 2025.

[2] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei, and J. Wen, "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, p. 186345, Mar. 2024.

[3] S. Kusal, S. Patil, J. Choudrie, K. Kotecha, S. Mishra, and A. Abraham, "AI-Based Conversational Agents: A Scoping Review From Technologies to Future Directions," *IEEE Access*, vol. 10, pp. 92337–92356, 2022.

[4] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, J. Bouwman, A. J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C. T. Evelo, R. Finkers, A. Gonzalez-Beltran, A. J. Gray, P. Groth, C. Goble, J. S. Grethe, J. Heringa, P. A. 't Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S. J. Lusher, M. E. Martone, A. Mons, A. L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S.-A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M. A. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, and B. Mons, "The FAIR Guiding Principles for scientific data management and stewardship," *Scientific Data*, vol. 3, p. 160018, Mar. 2016.

[5] D4Science, "D4Science – Accelerating Collaboration and Innovation," 2025.

[6] M. Assante, L. Candela, D. Castelli, R. Cirillo, G. Coro, L. Frosini, L. Lelii, F. Mangiacrapa, P. Pagano, G. Panichi,

and F. Sinibaldi, "Enacting open science by D4Science," *Future Generation Computer Systems*, vol. 101, pp. 555–563, 2019.

[7] M. Assante, L. Candela, D. Castelli, R. Cirillo, G. Coro, A. Dell'Amico, L. Frosini, L. Lelii, M. Lettere, F. Mangiacrapa, P. Pagano, G. Panichi, T. Piccioli, and F. Sinibaldi, "Virtual research environments co-creation: The D4Science experience," *Concurrency and Computation: Practice and Experience*, vol. 35, no. 18, p. e6925, 2023.

[8] L. Candela, D. Castelli, and P. Pagano, "The D4Science Experience on Virtual Research Environment Development," *Computing in Science and Engineering*, vol. 25, no. 2, pp. 12–19, 2023.

[9] S. Gesing, M. Pierce, S. Marru, M. Zentner, K. Huff, S. Bradley, S. B. Cleveland, S. R. Brandt, R. Ramnath, K. Kee, M. Dahan, B. M. V. Martínez, W. C. Sepulveda, and J. J. S. Mondragón, "Science Gateways and AI/ML: How Can Gateway Concepts and Solutions Meet the Needs in Data Science?," in *Critical Infrastructure* (A. D. Pietro and J. Martì, eds.), ch. 4, Rijeka: IntechOpen, 2023.

[10] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large Language Models: A Survey," 2024.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," 2023.

[12] J. Ye, X. Chen, N. Xu, C. Zu, Z. Shao, S. Liu, Y. Cui, Z. Zhou, C. Gong, Y. Shen, J. Zhou, S. Chen, T. Gui, Q. Zhang, and X. Huang, "A Comprehensive Capability Analysis of GPT-3 and GPT-3.5 Series Models," 2023.

[13] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, and S. Altman et al., "GPT-4 Technical Report," 2024.

[14] Google, "Gemini." `https://gemini.google.com/app`.

[15] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, and A. Fan et al., "The Llama 3 Herd of Models," 2024.

[16] Anthropic, "Introducing the next generation of Claude." `https://www.anthropic.com/news/claude-3-family`.

[17] Cheshire Cat AI, "Cheshire Cat AI – Production ready AI agent framework," 2025.

[18] Cheshire Cat AI, "Cheshire Cat AI." `https://github.com/cheshire-cat-ai`, 2025.

[19] M. Assante, L. Candela, D. Castelli, R. Cirillo, G. Coro, L. Frosini, L. Lelii, F. Mangiacrapa, V. Marioli, P. Pagano, G. Panichi, C. Perciante, and F. Sinibaldi, "The gCube system: Delivering Virtual Research Environments as-a-Service," *Future Generation Computer Systems*, vol. 95, pp. 445–453, 2019.

[20] Sebastián Ramírez, "Typer." `https://typer.tiangolo.com`.

[21] L. Boonstra, *Prompt Engineering*. Google, 2024.