Computer Science and Information Systems 23(1):419–441 https://doi.org/10.2298/CSIS250320078H

Towards Enhancing Data Science Agents with Semantics*

Sayed Hoseini[1], Maximilian Ibbels[1], Maximilian Knoll[1], and Christoph Quix[1,2,**]

[1] Hochschule Niederrhein University of Applied Sciences, Krefeld, Germany
sayed.hoseini@hsnr.de
[2] Fraunhofer Institute for Applied Information Technology FIT, St. Augustin, Germany
christoph.quix@fit.fraunhofer.de

Abstract. Data lakes, initially designed for storing heterogeneous datasets, have recently been extended with ML capabilities to unify data science tasks within a single platform. However, they still lack essential ML-specific features, limiting their effectiveness for end-to-end automation. Automated Machine Learning (AutoML) and Large Language Models (LLMs) offer potential solutions by streamlining various stages of the ML pipeline, yet both have significant limitations.
This paper presents an integration of AutoML frameworks and LLMs within a data lake system. We introduce a metadata model to capture data analytics processes, a Python package wrapping existing AutoML libraries, and a module utilizing LLMs to automate ML tasks. A comparative evaluation indicates that AutoML simplifies pipeline creation but limits user control and lacks robust data preprocessing support. LLMs can automate individual tasks, such as code generation, but struggle to orchestrate complete workflows effectively. Both approaches risk staying as basic prototypes that still need manual improvement.
The primary challenge lies in managing task interdependencies within ML pipelines. Retrieval-augmented generation enables dynamic access to external information but may overlook structured data relationships, leading to incomplete or redundant results. Therefore, we propose an extended vision that integrates multi-agent frameworks for data science with knowledge graphs that capture historical experience from previous ML experiments. We present preliminary results for developing comprehensive, context-aware ML agents and their integration into our data lake system SEDAR.

Keywords: AutoML, LLMs, Semantic Data Lake, MLOps, Data Science Agents.

1. Introduction

As modern data lakes incorporate an ever-growing variety of heterogeneous data sources, the complexity of utilizing this data also increases. The challenge lies in finding suitable methods to analyze these diverse and often unstructured data [36]. Traditionally, profound knowledge of data science methods is required to develop appropriate machine learning (ML) pipelines to optimally process data. Data science workflows are inherently complex, consisting of interdependent tasks like data processing, feature engineering, and model training. Addressing these tasks demands iterative refinement and real-time adjustments, as both the data and requirements are constantly evolving. This poses a significant

* This is an extended version of [20].
** Corresponding author

hurdle, particularly in domain-specific areas like chemistry, where experts often lack the necessary skills. Democratizing access to advanced analytics to a wider audience is thus highly desirable.

AutoML is the process of automating the end-to-end application of ML to real-world problems. However, existing AutoML approaches, reach their limits when it comes to automatically creating data-driven models with sufficiently high accuracy [26,20]. Furthermore, they focus on model selection and hyperparameter tuning, but do not help with other critical steps like data integration, processing and cleaning [42].

Large Language Models (LLMs) can potentially increase ML pipeline automation by assisting at all stages of the process. They can help with answering specific questions or even generate and execute the necessary code entirely and autonomously [14,38]. However, LLMs have difficulty understanding and adapting to specific requirements and contexts leading to generic, erroneous or inappropriate code for specialized tasks [20,25]. Furthermore, while they can be applied to more individual data science tasks [13,31], they still cannot perform holistic workflows and struggle to execute complete data science processes in one go. This is due to an inability to handle real-time changes in intermediate data and adapt dynamically to evolving task dependencies.

Both directions struggle to move beyond being merely used as an initial prototype that requires to be improved by humans manually to achieve the desired performance. Another weakness of these systems is that they start from scratch with each new model generation, without drawing on existing experience or knowledge [46]. Integrating the experience of historical training results, consisting of hyperparameters and various metrics related to result quality, can significantly accelerate and improve this process.

Recently, several techniques have been proposed to tackle these challenges with LLMs. One of the most notable is Retrieval-Augmented Generation (RAG), which leverages information retrieval methods to identify data relevant to the question at hand. Relevant information can then be provided to the LLM as additional context during inference, often leading to improved answer quality [12]. Another promising approach is combining LLMs (and potentially RAG) with knowledge graphs (KGs) [17]. Knowledge graphs use graph-based representations to organize, integrate, query, and reason about diverse collections of data and knowledge in a structured manner.

In this article, we address the question of how the integration of automated machine learning can augment the capabilities of a data lake system and compare the efficacy of AutoML tools with LLMs in this context. In addition, we examine the question of how such general-purpose AI systems can be customized and optimized for data science activities by integrating knowledge from structured KGs. Our contributions include:

1. We propose an integrated framework *AutoMLWrapper* for heterogeneous AutoML systems, introducing a uniform abstraction layer for job orchestration.
2. We introduce a methodology for LLM-assisted configuration and automation of ML pipelines in data lake environments. The approach combines generative reasoning with structured metadata management to improve data science workflows.
3. We present a conceptual framework that unifies RAG, KGs, and LLM-based data science agents, highlighting their complementary roles: RAG for contextual retrieval, KGs for semantic grounding, and LLMs for adaptive reasoning and synthesis.

4. We investigate the semantic integration of LLMs with the OpenML knowledge graph, aiming to exploit metadata on prior ML experiments for the development of semantics-aware data science agents.
5. To validate the proposed concepts, we implemented a prototype framework within our *SEDAR* data lake system [16]. The prototype operationalizes the integration of AutoML components and LLM-based agents, demonstrating the feasibility and practical relevance of the approach.
6. We empirically evaluate the proposed framework on complex, non-standard ML tasks in the domain of chemistry, demonstrating its applicability and discussing the relative performance and interpretability of LLM-based and AutoML-based techniques.

This paper is an extended version of [20], in which we presented a preliminary version of our framework, in particular the *AutoMLWrapper* and its comparison with LLM-generated ML pipelines. These results are presented in section 3. Compared to [20], we have updated the discussion of related work in Section 2, to include recent approaches on AutoML, LLMs and their application as data science agents. Section 4 is an original contribution of this paper. It proposes an architecture for a KG-enhanced data science agent for more comprehensive ML workflows. The evaluation in Section 5 has also been extended to include results from the KG-enhanced approach.

2. Related Work

During his presentation at the NeurIPS 2024 conference, Ilya Sutskever – one of the researchers behind ResNets [29] and co-founder of OpenAI – argued that the era of pre-training neural networks, which has primarily focused on scaling compute and data, has reached a limit [39]. While computational power continues to grow through larger clusters and advancements in hardware and algorithms, the data side has plateaued, as the global Internet, essentially the only vast data resource, is already more or less fully utilized. From this saturation of data, he argues, emerges the increasing importance of autonomous agents moving forward. In the context of AI/ML, agents typically refer to autonomous systems that can take actions, make decisions, and interact with their environment to achieve specific goals [10]. LLM-based agents for data science (short: data science agents) activities essentially resolve to instruct a "large" AI to create "smaller" ones.

Early works on LLMs for AutoML Tornede et al. [40] review the current state of the integration between LLMs and AutoML in both directions. Currently, interacting with an AutoML system is often still challenging for non-expert users. They highlight the potential to substantially improve the human interaction component of AutoML tools, alleviate the tedious task of correctly configuring the tool which often requires experts, assist in interpreting the output, and improve several internal components through knowledge. Simultaneously, the integration also bears risks such as inadequate evaluation and catastrophically wrong behavior of AutoML systems due to hallucinations [25] of an LLM-powered component, and ever-increasing resource demands.

Sun et al. [37] present a survey on LLM-based data science agents including a timeline of publications, separated into commercial and publicly available ones. They find

major obstacles to be the integration of domain-specific knowledge and handling multi-modal inputs, i.e. user instructions alongside charts, tables, code, etc. as well as planning complete data workflows (see below).

An early framework that uses *ChatGPT* as a *Virtual Interactive Data Scientist* is presented in [11]. The research represents a systematic approach to creating a personal data scientist, but the authors admit that there are still severe problems to be addressed, such as erroneous recognition of the correct current state by the LLM.

Several works utilize LLMs for hyper-parameter optimization [51,50,32,35]. These methods help with model selection and allow an agent to conduct experiments with specific hyper-parameters for selected algorithms to iteratively optimize them based on historical trials while autonomously processing the task information. Chen et al. [2] integrate LLMs as adaptive operators in an evolutionary neural architecture search, Wei et al. [43] deploy LLM-based agents to devise tailored solutions for diverse graph-structured data and learning tasks. In [23,8] we find a novel framework for benchmarking AI research agents. *AutoMMLab* [45] is an LLM-empowered AutoML system that follows the user's language instructions to automate model production workflows for computer vision tasks. Other methods focus on feature engineering [13,30,33].

Microsoft's *CoML* [50] (formerly MLCopilot) promises to derive knowledge in textual form, which can be reused for new machine learning requirements. The idea is to perform knowledge-based reasoning, that is to leverage LLMs to conduct reasoning and task solving based on previous knowledge. This knowledge is extracted from historical data and elicited into an experience pool in text form that is provided to the LLM at runtime for a given task. However, it relies on LLM-generated embeddings of task descriptions and the prototype[3] can only return hyperparameters for a predefined method.

End-To-End Data Science Agents All of the aforementioned methods only focus on substeps in the ML pipeline. A major challenge is the complexity of planning the entire pipeline, mainly due to the interdependencies between each step. For example, the type of dataset influences preprocessing and neural network design, which in turn affects the hyperparameters that need optimization for a specific downstream task. These inter-step dependencies expand the search space to account for all possible combinations of related steps. Additionally, supporting multiple downstream tasks further complicates the process, as each task has its specific requirements [41,14]. For example, data cleaning and feature engineering are essential prerequisites, but they also must be tailored to each case.

The *Data Interpreter* [14], part of the multi-agent framework *MetaGPT* [15], first constructs a dynamic plan graph (*task graph*), which is used to capture dependent tasks and arrange them in a meaningful sequence. For each of the seven tasks (data exploration, correlation analysis, outlier detection, feature engineering, model training, model evaluation, and visualization) the LLM creates an *action graph* to execute actions in the form of generated code. Those are executed locally to obtain partial results. Errors are sent back to the LLM for autonomous debugging. Additional customized scripts can be provided as tools, for example, for data exploration/analysis.

DS-Agent [9] integrates LLMs with Case-Based Reasoning (CBR) to automate data science tasks. In the development stage, it structures an automatic iteration pipeline using the CBR framework to leverage expert knowledge from platforms like Kaggle and facili-

[3] https://github.com/microsoft/CoML

tates consistent performance improvement through a feedback mechanism. This paradigm adapts past successful solutions for direct code generation, which significantly reduces the demand for the foundational capabilities of LLMs, enabling efficient deployment with lower resource requirements. As opposed to RAG, which focuses on retrieving and integrating real-time information from external data sources to improve the model's response, CBR focuses on case libraries that store detailed solutions to specific problems, which are reused and adapted for new situations.

AutoML-Agent [41], a multi-agent framework takes user's task descriptions and implements collaboration between specialized LLM agents to deliver deployment-ready models. Five different agents interact with each other each responsible for a different task, in particular managing prompts, data, models, operations and agents. The framework includes agent specifications, a prompt parsing module, a retrieval-augmented planning strategy, a prompting-based plan execution, and a multi-stage verification. Experiments on seven ML tasks demonstrate that *AutoML-Agent* outperforms existing methods in terms of success rate and downstream task performance.

AgentKv1.0 [7] manages the data science lifecycle through reasoning frameworks that dynamically process memory using past experiences to guide future decisions. The system integrates various tools and libraries like *Torchvision* and *HuggingFace* to handle multimodal data from computer vision, natural language processing, and tabular data. The agent is evaluated through a competitive *Kaggle*[4] benchmark. The evaluation framework assesses end-to-end capabilities, from task retrieval to submission on *Kaggle* and achieved a 92.5% success rate in automating tasks, spanning multiple domains and data types. The agent ranks in the top 38% of 5,856 human Kaggle competitors, performing at a skill level comparable to Expert-level users (Grandmaster level on *Kaggle*). Notably, *AgentK* has won 6 gold medals, 3 silver medals, and 7 bronze medals in *Kaggle* competitions. Unlike traditional LLM methods that rely on chains of thought, *AgentK* employs a memory module for continuous improvement and adaptation that allows for a flexible learning paradigm, where the agent can generalize across different tasks by leveraging past experiences.

To summarize, LLMs still work relatively well on classical tasks like tabular classification and regression as compared to traditional AutoML libraries, such as AutoGluon, but LLM-based agents work significantly better in complex tasks. However, tasks such as reinforcement learning and recommendation systems pose particular challenges in both paradigms. On one hand, existing frameworks place less emphasis on these areas, and on the other, contemporary LLMs have received less specialized training to effectively address them. In terms of limitations, even though these agents offer increased flexibility across ML tasks and data modalities, the absence of skeleton code increases the risk of code hallucination. Additionally, the code generation quality varies heavily when using different backbones, e.g. GPT or open-source models. Developing a more robust framework less reliant on the LLM backbone is imperative [41].

[4] https://www.kaggle.com/

3. Enhancing the Machine Learning Capabilities in SEDAR with AutoML and LLMs

This section is largely based on our article [20]. We use the resulting findings to motivate researching semantics-aware agents for data science tasks.

3.1. Integrating AutoML and LLMs with *SEDAR*

SEDAR is designed in a modular, easily extendable, and flexible six-layer architecture based on open-source technologies deployed as microservices (see **??**). To realize a true polystore [27], the *storage layer* is composed of different storage systems to support efficient storage for heterogeneous data structures based on their specific data model. By default, *Hadoop* with its file system *HDFS* is used to store files with arbitrary structures including audio, images, video, text, and ML model artifacts. The *transformation layer* implements various operations for data preparation, e.g., cleansing, transformations, or joins. We employ *Apache Spark* as a unified data transformation engine for large-scale data processing. *Spark*'s engine can process arbitrary data sources in the form of *DataFrames*. A programmatic interface to communicate with *SEDAR* is provided by the *SEDAR-API*, an external Python module. To ensure generality, the *ingestion layer* is independent of any formats and data sources, not limited to particular systems in the storage layer, and allows for versioning and the addition of further databases. The metadata service extracts and manages information about versions, updates, profiles, etc. automatically. The *interaction layer* facilitates comprehensive metadata management while ensuring ease of use for non-technical users.

One of *SEDAR*'s core objectives is to facilitate knowledge management on top of the stored data. Semantic data can enrich basic metadata, such as schema and data types, by adding context information not originally present in the data source. One way to achieve this is to map raw data from heterogeneous data sources to semantically rich models to increase the usability and interpretability of data. For this, a *semantic data management layer* [21] has been incorporated that allows to associate semantics to datasets. Using these semantics, *SEDAR*'s semantic query engine allows querying heterogeneous data sources using a uniform semantic data model via ontology-based data access [21].

SEDAR is deployed in Industry 4.0 settings [49] in the chemical domain [**?**,20], which increasingly demands modern data lakes to aid in creating, managing, and deploying ML applications while also providing necessary data [5]. To facilitate Machine Learning Operations (MLOps) [28], in a *data analytics layer SEDAR* utilizes MLFlow [47], an open-source framework to manage the inputs and outputs of ML applications [47]. This includes tracking model versions, code, and tuning parameters, ensuring reproducibility of results, and facilitating model deployment. **??** illustrates the general view of the machine learning tab that lists existing experiments and deployed models at the top and shows details about past runs associated with an experiment at the bottom. *Jupyter Notebooks* have been integrated into the interaction layer and are hosted from a *JupyterHub*, allowing users to have personalized work environments. Authentication between web services is centralized using the OAuth protocol and JWT tokens. Our self-hosted GitLab instance serves as the identity provider and central repository for storing code. Within *SEDAR*,

users can select datasets, and then choose the desired learning method, and the system will automatically generate a notebook with the necessary code to load the *DataFrames* from storage via the *SEDAR API*, register their experiments, track individual runs and software environments, store model artifacts, and deploy their applications.

3.2. AutoMLWrapper

The *AutoMLWrapper* is a stand-alone Python module acting as an interface for three open-source AutoML frameworks: *AutoKeras*, *AutoGluon* and *AutoSklearn*[5]. Special emphasis is placed on the standardized construction of AutoML jobs by configuring the underlying libraries using a common set of parameters. Object-oriented programming for library-unspecific logic facilitates easy integration of new AutoML libraries without redesigning the entire interface. To use the package with code as well as in combination with *SEDAR*'s frontend, presenting information about possible settings for AutoML experiments required a form that can be easily understood and displayed. To meet these requirements, each of the three frameworks is equipped with a configuration file essential for the correct call to the underlying library. Then, there is a global configuration file that contains meta-information about the available libraries' scope and assigns data and problem types. For example, the file can be used to decide in the frontend which (library) functions are available to the user for image segmentation.

The *AutoMLWrapper* is integrated into *SEDAR* within a dedicated virtual Python environment built into the *Jupyterhub*'s default image. Similar to the previous manual training scripts a complete *Jupyter Notebook* is created and populated with information from the frontend. By design, the package also integrates with *MLFlow* to manage experiments and save the generated models. Despite the idea of automation, the package also provides enough freedom for experienced users allowing them to add their logic and specific customizations. This notebook serves as the basis for running the AutoML job, of which relevant metrics and parameters are automatically recorded and tracked by *MLFlow*.

3.3. LLM-Orchestrated ML

LLMs assist the user at multiple locations in the *SEDAR* framework. A new user interface based on *LangChain* has been implemented to enable communication between the user and LLM providers, allowing for the dynamic integration of additional information from the system's metadata. The current implementation offers a simple chat functionality from the frontend. The interface can be used with various LLM providers.

JupyterAI is an extension for *Jupyter* that enables the use of LLMs within a separate chat window and the cells of a notebook. It distinguishes between language and embedding models. The embedding model is used to create a compressed representation of the contents of a particular directory, and the language model is used to ask specific questions about the file content. Users can either have methods or code from the notebook explained to them, or have code sections and entire notebooks generated. *JupyterAI* can be connected to a range of default models, but also to *Hugging Face*, *GPT4All* and *Ollama* which include a variety of open-source LLMs of varying sizes.

[5] github.com/automl/auto-sklearnl/keras-team/autokerasl/autogluon/autogluon

Furthermore, the *CAAFE* library [13] is used to automate feature engineering of classification problems with tabular data using LLMs. The library has been extended to image data in classification and segmentation problems. *CAAFE* processes a dataset D in an iterative process. An LLM is dynamically prompted to generate code. By executing this code on D, a new dataset D' is created, which contains the feature changes of this iteration. An identical model is trained for both datasets and problem-dependent metrics are measured. These metrics are communicated to the LLM for the next iteration, which receives D' in the next iteration if there is an improvement.

4. Unifying Knowledge Graphs with Data Science Agents

Building on the insights from related work, it becomes evident that, despite their transformative potential, LLMs still face several fundamental challenges that limit their reliability and applicability. According to Hogan et al. [12], these challenges include the tendency of LLMs to generate information that is not grounded in factual data (often referred to as hallucination), the lack of transparency regarding the sources and reasoning behind their outputs (opaqueness), the difficulty of maintaining up-to-date knowledge due to the high cost of retraining (staleness), and the inherent inability to provide exhaustive or complete responses as a consequence of their probabilistic nature (incompleteness).

As we have seen in Section 2, to address these issues, data science agents often rely on RAG mechanisms. RAG operates by combining the generative capabilities of LLMs with a retrieval mechanism that actively searches for relevant information. The retrieval process acts as a bridge to external knowledge, allowing the agent to pull in additional information that might not be delivered from the model's internal parameters only. This external data is then used by the LLM to generate more accurate and context-aware responses. In the context of data science, this can be historical experience, i.e. prior solutions to similar problems as seen in frameworks like *DS-Agent*, *AutoML-Agent*, and *AgentK* [9,41,7] (see Section 2), where retrieval is used to ensure that the models can dynamically adjust to new, unseen problems by leveraging a pool of knowledge about past experiences.

The connection to search engines (SE) and information retrieval (IR) systems is key here. Just as search engines index and retrieve the most relevant documents based on user queries, RAG enables LLMs to access and integrate external data sources. This improves their ability to provide solutions to complex data science tasks, particularly when dealing with multimodal data, and when addressing issues that require dynamic knowledge updates. While search engines excel in providing broad coverage and real-time updates, with transparent and efficient access to vast amounts of data, they struggle with returning complete and precise answers due to noisy or incomplete data, and they cannot generate or synthesize new content from their indexed sources [12]. In addition, Peng et al. [34] identify several limitations inherent to current Retrieval-Augmented Generation (RAG) approaches. First, conventional RAG methods often fail to capture the structured relationships that exist among pieces of textual information. While semantic similarity enables retrieval of contextually relevant content, it overlooks relational knowledge such as citation links or entity associations that are essential for a comprehensive understanding of interconnected data. Second, RAG systems tend to introduce redundancy when retrieved documents are concatenated into prompts, resulting in unnecessarily lengthy and repetitive inputs. Finally, RAG architectures frequently lack mechanisms to integrate informa-

tion at a global level: although relevant snippets can be retrieved from multiple sources, these fragments are not always synthesized into a coherent or contextually consistent representation of the underlying knowledge.

In contrast, knowledge graphs offer highly curated, structured data that ensures more precise, complete, and consistent results, especially in domain-specific contexts [6]. They explicitly store rich factual, interconnected knowledge, providing a structured, semantically rich representation of domain-specific relationships and entities, enabling more precise and context-aware retrieval of relevant information. However, they are difficult to construct and evolve by nature, making it challenging to generate new facts and represent unseen knowledge [24]. As previously described, LLMs have significant limitations due to the lack of external knowledge [1]. So-called *KG-enhanced LLMs* integrate knowledge graphs with LLMs to address the difficulty in recalling factual knowledge by aiming to generate knowledge-grounded contents [22]. An example is *KG-Rank* [44], where the authors leverage a medical knowledge graph with ranking techniques, aiming to improve free-text question-answering in the medical domain. Upon receiving a question, they initially retrieve factual information from a medical knowledge graph and then apply a ranking to obtain the most relevant triplets which then, combined with the task prompt, are input into LLMs for answer generation. A similar idea is presented by *CoML* [50], where, although it is not powered by knowledge graphs, the LLM is assisted by a so-called knowledge pool to retrieve relevant experience based on the task description. In

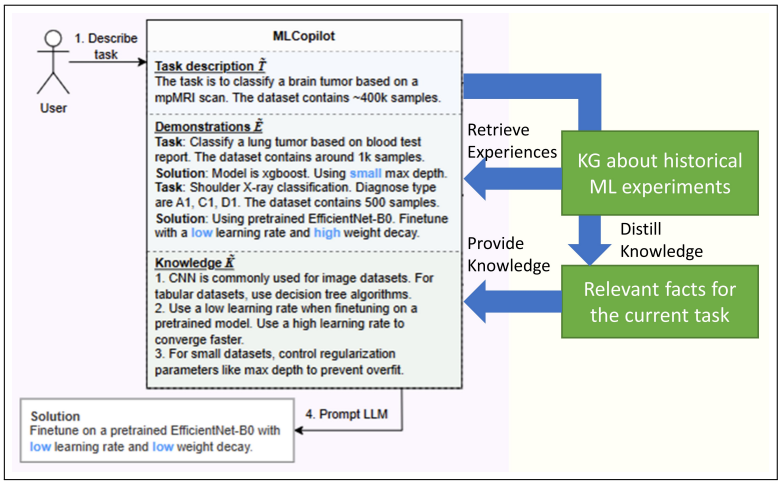

Fig. 1. Example of a KG-enhanced data science agent, extended from *CoML* [50]

contrast, Figure 1 applies this idea to knowledge graphs and shows how factual knowledge about past ML experiments can potentially be distilled and provided to an LLM for automated data science. The performance is improved by adapting dynamically to historical data relevant to the task at hand. Importantly, in this example, both RAG and knowledge graphs are used in synergy. While RAG provides dynamic access to external data sources, the knowledge graph ensures that the retrieved information is structured, precise, and con-

textually relevant. Therefore, it is reasonable to view not only search engines and LLMs but also knowledge graphs as complementary technologies whose integration has the potential to produce synergy, capitalizing on the strengths of each while mitigating their respective weaknesses. **??** provides a comparison between the discussed technologies, LLMs, search engines, knowledge graphs as presented by Hogan et al. [12].

Search engines excel at providing broad coverage of diverse topics quickly, but their results may be imprecise or incomplete due to noisy data. They rely on indexed content and cannot generate or synthesize new information. Knowledge graphs offer highly structured, domain-specific data, ensuring precise and complete results, but their coverage is narrower compared to search engines, and they require more complex querying. LLMs are versatile and generative, capable of creating novel content and synthesizing data from multiple sources, but they may produce hallucinations, generating incomplete or inaccurate results. While search engines and knowledge graphs are more transparent in terms of the data used, LLMs are often opaque, making it harder to understand how answers are derived. Search engines and LLMs are user-friendly with natural language input, whereas knowledge graphs are typically harder to use due to their reliance on structured queries. Efficiency-wise, search engines and knowledge graphs handle queries efficiently, while LLMs require more resources.

4.1. Towards a KG-enhanced Data Science Agent

In the following, we describe current research directions in our lab examining the question of how to extend the well-known integration of search engines and LLMs by a component that also leverages knowledge graphs.

We have seen how information retrieval as well as knowledge graphs can enhance the performance of LLMs. While RAG-based information retrieval has certain advantages, for data science tasks it is necessary to provide accurate factual knowledge capturing the interconnection between multiple resources, for which only knowledge graphs represent viable solutions. The essential role of knowledge provider will be covered by *MLSea* [4] in our approach.

In *CoML* [50], the source of historical data is not clearly defined. In contrast, *MLSea* aggregates ML datasets, experiments, software, and scientific works from *OpenML*, *Kaggle*, and *Papers with Code* into a structured knowledge graph. Key components of *MLSea* include the *MLSO* ontology, *MLST* taxonomies, and the *MLSea-KG* knowledge graph, which collectively aims to improve the search, explainability, and reproducibility of ML pipelines. *MLSea* thus provides a unified view of ML processes, helping users discover datasets, models, algorithms, hyperparameters, and relevant publications in one place. This system may address the challenges of accessing diverse ML knowledge and support research in automated ML systems and experiment documentation. To effectively apply RAG to the *MLSea* knowledge graph, it is essential to first understand the underlying structure of the graph. Adapting the retrieval mechanism to the knowledge graph's data model — such as the types of entities (e.g., algorithms, models, tasks) and their connections (to datasets, hyperparameters, runs, etc.) — ensures the ability to accurately query the graph for relevant information.

When exploring *MLSea*, we quickly observed a strong overall heterogeneity, i.e., varying quality of descriptions, incompleteness in different patches of the graph, heterogeneous notation, etc. Here, *AssistML* [46] can provide valuable assistance. It proposes a

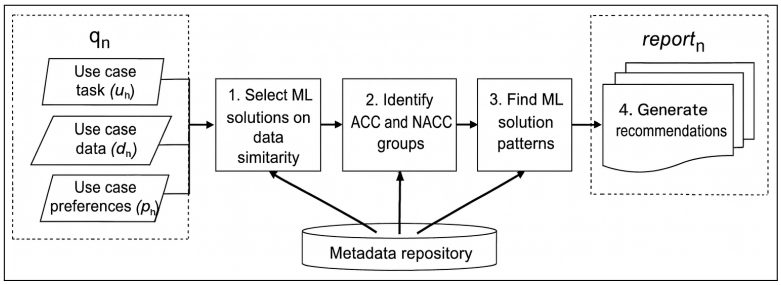

Fig. 2. Overview of the workflow of AssistML [46]

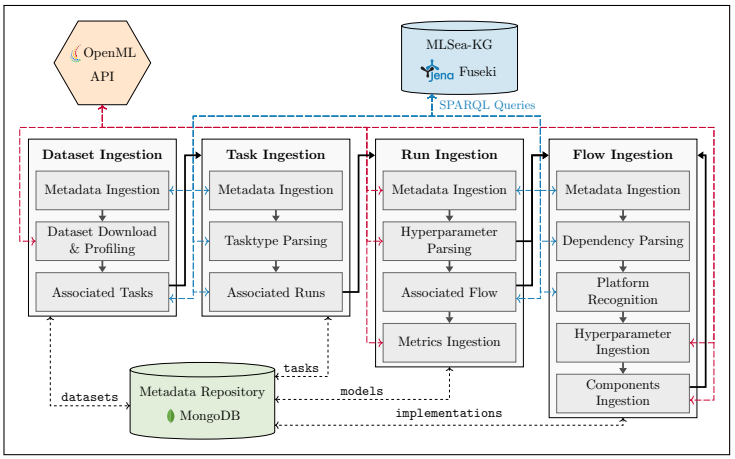

Fig. 3. Ingestion pipeline

metadata model that standardizes different datasets and task types, making them comparable and enabling it to suggest solutions for target metrics without relying on LLMs. As shown in fig. 2, its original workflow is divided into four steps: (1) select ML solutions based on data similarity, (2) identify acceptable (ACC) & nearly acceptable (NACC) ML solutions, (3) find ML solution patterns and finally (4) generate a list of recommendations. Our idea is to populate metadata repository with data from MLSea.

For this, we extended this implement the ingestion pipeline shown in Figure 3. We decided to limit the implementation to data related to *OpenML* to decrease complexity because the bulk of data is stored here and the remaining two subgraphs show higher incompleteness and heterogeneity. The ingestion extracts, transforms, and loads *OpenML Datasets*, *Tasks*, *Flows*, and *Runs*[6] into a structured framework.

Each dataset is analyzed using a data profiler to understand its characteristics, which are then stored in a *MongoDB*. Once datasets are processed, the pipeline retrieves and processes ML tasks linked to them, such as classification, regression, clustering. Task data establishes relationships between tasks and their corresponding datasets. The pipeline then processes implementations by extracting metadata, including dependencies and soft-

[6] https://docs.openml.org/concepts/, accessed on 11.03.2025

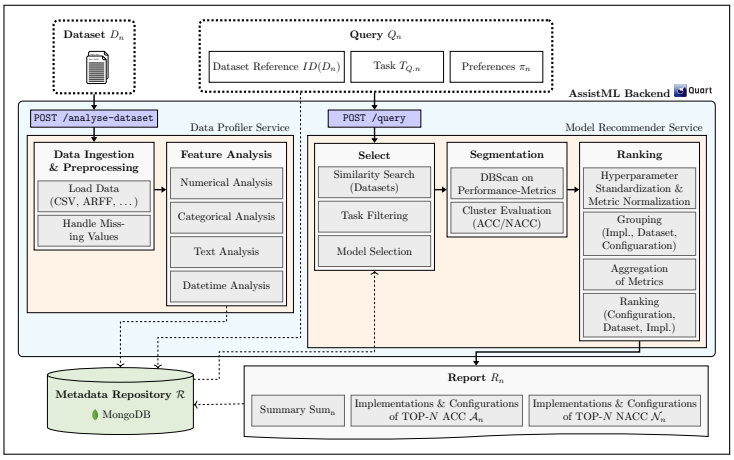

Fig. 4. Backend for generating reports

ware requirements, from *OpenML*. Implementations are then stored in the database and linked to relevant tasks. For model processing, the pipeline retrieves experimental runs associated with tasks and extracts model performance data. It constructs model setups by combining hyperparameters, implementations, and tasks to create a complete experiment configuration. Evaluation metrics are derived from run data, enabling performance assessment and comparison.

MLSea serves as a semantic layer that connects datasets, tasks, implementations, and models using well-defined relationships. SPARQL queries help retrieve dependencies and software requirements, which are structured as linked entities in the knowledge graph. This way, the pipeline can dynamically fetch available implementations and cross-check them with *OpenML* before storing them. The knowledge graph also plays a role in model processing to retrieve existing experiment configurations, hyperparameters, and performance metrics. The structured representation ensures that models are linked to their corresponding datasets, tasks, and implementations. This enables efficient retrieval of past experiments for knowledge-driven recommendations providing a structured foundation for AutoML workflows.

Once the metadata repository is populated with the historical data, the backend, shown in Figure 4 can generate reports. This means for a given query, composed of a dataset, a task and user preferences, it generates model recommendations. The pipeline processes the dataset first using the data profiler, identifying feature types and target attributes. The model recommender is designed to suggest models with hyperparameters and configurations based on dataset characteristics and user-defined performance criteria. The module leverages the metadata from existing ML solutions to facilitate the rapid identification of suitable models, thereby streamlining the model selection process for users. Users can specify performance preferences, such as accuracy, training time, or resource consumption and the recommender system aligns its suggestions with these criteria. This efficiency contrasts with traditional AutoML systems, which may require extensive computational resources and time to explore various model configurations. The system offers explana-

tions for its recommendations, making it accessible to users with varying levels of ML expertise.

The output of the recommender component is a report, a structured evaluation for a given query, which provides a summary of the best models, their scores, and explanations regarding why they were chosen. In the report, models are ranked according to multiple factors, including dataset similarity, implementation characteristics, and metric performance, to provide recommendations on the most suitable models for a given dataset and task. In the final stage of the workflow (see Figure 5), models from the report are included in a prompt that are passed to a LLM to generate executable ML pipeline code. The produced code is then executed in the local environment on the specified dataset, and evaluation metrics are computed. The results and metrics are returned as output, providing an empirical assessment of the generated pipeline.

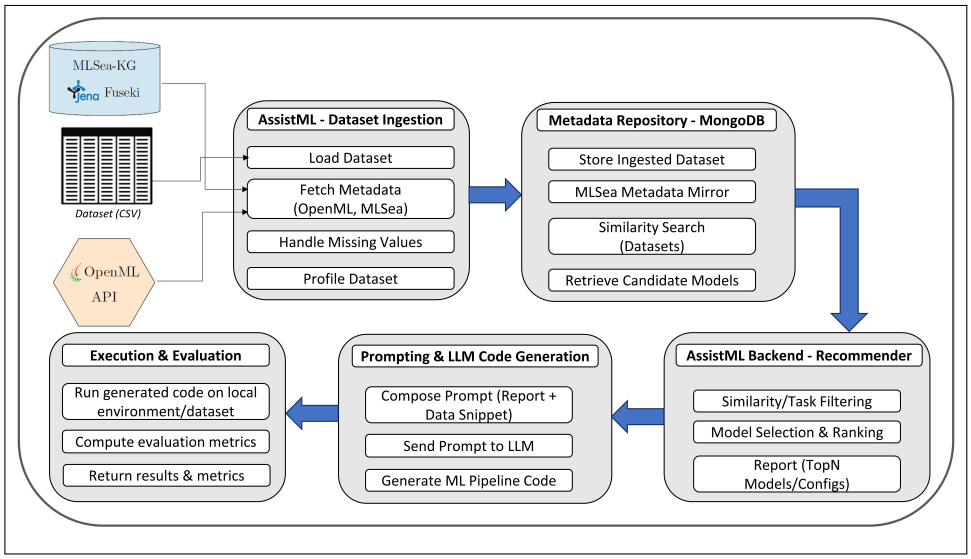

Fig. 5. Code generation pipeline with LLMs and MLSea-enhanced AssistML recommendations

5. Evaluation

5.1. Evaluation Setup, Datasets, and Criteria

We evaluate the proposed frameworks against problems from real-world use cases at the Institute for Surface Technology of the Niederrhein University of Applied Sciences (HIT). The HIT researches (organic) coatings and paints which play a vital role in safeguarding surfaces. ML is applied in this context for experiment suggestions and data analysis. All datasets, code, and training details for this study have been exported from *SEDAR* and made available online. For detailed information about dataset sizes and class distributions,

we refer to the corresponding publication, as we have tried to reproduce the experimental settings accordingly as much as possible. The experiments are performed in the described *Jupyter* environment equipped with a *Nvidia Quadro RTX8000* graphics card, two *Intel Xeon Gold 6230R* processors, and 754 GB RAM. Training for all models is initially limited to one hour. For the *AutoKeras* library, for which no time limit can be defined, a division into 50 models for 50 epochs is used.

Because working with LLMs can be freely designed by the user, rules to improve reproducibility are defined. LLMs in *GPT4All* are designed for the English language, hence the conversations are conducted in English. Problems are described by the user. It is assumed that the user cannot provide an exact definition of the sequence of experiments due to a lack of experience. Hence, ideally, the user does not use any ML terminology when working with LLMs. For example, it should not be mentioned that the class distribution of the data set is unbalanced. Instead, it should be checked whether the contextual information provided by the *SEDAR* AutoML solution is sufficient for the LLM to act accordingly. Any further steps identified by the LLM through continued querying for a more robust ML pipeline are tracked as further effort. The two open-source LLMs that are available in

```
%%ai chatgpt Generate a Python code snippet for an Image Segmentation pipeline using the libraries Tensorflow or Pytorch and other
appropriate libraries, optimized for GPU usage. The code should handle a dataframe called gitter_trainv1 with columns 'image' and
'label'. 'image' contains the path to the image file and 'label' contains the path to the mask file. The dataset contains 348 images.
All images are 256px in height, 256px in width, in GRAY mode, and saved as png. Focus on a simple yet effective neural network architecture
suitable for beginners. You may use a pre-trained model and fine-tune it on the dataset. Include necessary imports, dataset loading and
preprocessing steps, model architecture, and training loop. The code should be ready to run in a Python interpreter without modifications.
datagen = ImageDataGenerator(rescale=1./255)
image_generator = datagen.flow_from_dataframe(dataframe=gitter_trainv1, x_col='image', y_col='label',
                                              target_size=(256, 256), color_mode='grayscale',
                                              class_mode='input', batch_size=32)
base_model = MobileNetV2(input_shape=(256, 256, 1), include_top=False)

x = base_model.output
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

model = Model(inputs=base_model.input, outputs=x)
model.compile(optimizer=Adam(), loss='binary_crossentropy')
model.fit(image_generator, epochs=10)
```

Fig. 6. Example prompt using *Jupyter Magic* with *GPT4*'s response to generate the necessary code for solving the image segmentation task. Shortened for brevity

JupyterAI through *GPT4All nous-hermes-llama2-13b* (Hermes) and *mistral-7b-openorca* (Orca) are used by default, as two representatives with varying parameter sizes and instruction and chat model respectively. Of these, we include the best-performing model in the evaluation. The experiment begins with a single prompt (Figure 6) to solve the problem in one shot. If this prompt does not create a valid ML pipeline, e.g., due to errors in the created code, the prompt is modified, for example, to split the problem into several prompts. In this case, it is tracked to what extent additional effort is required from the user and how demanding the changes are. If no approach with open-source LLMs leads to success, *GPT4* is used.

5.2. Datasets

The first dataset [18] contains images of four different types of surface defects. The images were collected via Laser-scanning microscopy to obtain height profiles on the surfaces and produce an artificial RGB image by a threefold repetition of the height data. This data is evaluated for **object detection**. While the authors dealt with Few-Shot Object Detection [3], these solutions are not included in the AutoML libraries, and except for *GPT*4, none of the LLMs had information about such models. Hence, we compare to the *RetinaNet* baseline given in the original paper [18].

For the same dataset [18] of surface defects optical images are available for **image classification**. Here we limit the number of available training samples per class to simulate a Few-Shot setting.

The next dataset [48] consists of images of scratch tests on surfaces to be evaluated for **image segmentation**. Zhang et al. identify the area where the so-called cross-cut test has removed the surface material. A binary mask with the marked regions is available for each image. They use several image augmentations to increase the amount of training data and make the model more robust. For this dataset, only the original images are available, without augmentations. However, feature engineering using *CAAFE* is performed.

The last dataset comes from [19], where methods for analyzing the resistance of coatings and paints are discussed again. The breakthrough point of a needle on a coating is identified by using sensor data about the movement and the force exerted by the needle on the surface. Using this data, the authors conduct **tabular regression** to find the position of the penetration through the surface. Based on the force exerted on the needle, they infer the scratch resistance of the evaluated coating.

Criteria A series of evaluation criteria are defined to conduct a structured evaluation of how effective the AutoML- & LLM-based approaches are in solving various ML tasks within the SEDAR system. If, for example, the effort is evaluated, more effort corresponds to a less favorable value, which is why an ascending color coding is selected (■■■■■■). If the completeness of a solution is being evaluated, the process is reversed; higher values are better (■■■■■■).

The performance evaluates the predictive capabilities of the models created by the AutoML solutions. Various metrics are considered, e.g. *Mean Average Precision* (MAP) for object detection, and *Intersection over Union* (IuO) for image segmentation. Where possible, the values are determined using the built-in functions of the AutoML library. If the appropriate metric is not implemented, this is highlighted negatively and a corresponding implementation is imported.

For efficiency, the time and resources required to experiment cover the process from data selection to the finished model. The time calculation in turn includes a) the preparations for the experiment (t_{prepare}), such as the dialogs, b) the time spent in the source code (t_{code}), for example by adapting function parameters, and c) the computing time required (t_{compute}). The complexity of the configuration of an experiment or the effort required for any changes to the code is measured by the following color-coded scale as *work*: (1) immediate execution, (2) simple, (3) medium, (4) extensive configuration and (5) individualized configuration with external resources.

The amount of prior knowledge and programming skills required for the configuration and changes to the code for a functioning pipeline is rated.

Finally, the initial model created by the AutoML process is assessed. A good starting situation either directly provides a versatile selection of models and parameters that the user can continue to use, or, especially when working with LLMs, provides sufficient hints to improve the designed pipeline.

5.3. Results of AutoMLWrapper and LLM-based Pipelines

Table 1. Overview of the experiments related to object detection. Only *AutoGluon* can perform object detection. The open-source models were unable to deal with this task, hence *GPT*4 had to be involved. Distribution of train/test sets: 140/36
More effort corresponds to a less favorable value, thus ascending color coding (🟩🟩🟨🟧🟥).
For completeness of a solution the process is reversed; higher values are better (⬛⬛⬛⬛⬛).

	$\text{Wrapper}_{\text{gluon}}$	LLM_{gpt4}	[18]
MAP	0.46	0.4	0.458
t_{prepare}(min.)	2	1	
t_{code}(min.)	< 1	10	
t_{compute}(min.)	60	12	
work	🟩□□□□	🟩🟩🟨□□	
Prior knowledge	🟩□□□□	🟩🟩□□□	
Programming expertise	🟩□□□□	🟩🟩🟨□□	
Initial model	⬛⬛⬛⬛□	⬛⬛⬛⬛⬛	

Table 2. Overview of the experiments in the segmentation data set. Only *AutoGluon* can perform image segmentation. Distribution of train/val/test sets: 174/35/8

	$\text{Wrapper}_{\text{gluon}}$	LLM_{Orca}	[48]
IoU	0.82	0.91	0.96
t_{prepare}(min.)	3	1	
t_{code}(min.)	3	15	
t_{compute}(min.)	9	20	
work	🟩🟩□□□	🟩🟩🟨🟧□	
Prior knowledge	🟩🟩🟨□□	🟩🟩🟨□□	
Programming expertise	🟩□□□□	🟩🟩🟨□□	
Initial model	⬛⬛⬛□□	⬛⬛⬛□□	

The results of the experiments (Tables 1 to 3) show that the *AutoMLWrapper* can considerably simplify creating ML pipelines. All results are from a single run. The user only has to make minimal entries in the frontend and can still create powerful models, which is particularly advantageous for users without much experience. The *AutoMLWrapper* can autonomously train and evaluate models in a short time if the appropriate settings are made. The performances in terms of evaluation metrics are competitive compared to the

Table 3. Overview of the image classification experiments with limited training data. *AutoGluon*'s few-shot setting leads to the best results. Distribution of train/test sets: first row 20/79 and second row 80/79

	$\text{Wrapper}_{\text{gluon}}$	$\text{Wrapper}^{\text{few-shot}}_{\text{gluon}}$	$\text{Wrapper}_{\text{keras}}$	$\text{LLM}_{\text{Hermes}}$
F1	0.62	0.87	0.6	0.5
F1	0.85	0.99	0.6	0.55
t_{prepare}(min.)	2	2	2	2
t_{code}(min.)	< 1	<1	15	10
t_{compute}(min.)	< 1	< 1	13	17
work	■□□□□	■□□□□	■■■□□	■■■■□
Prior knowledge	■□□□□	■□□□□	■■■□□	■■■■□
Programming expertise	■□□□□	■□□□□	■■□□□	■■□□□
Initial model	■■■■□	■□□□□	■■■□□	■■■■□

given baselines for both the *AutoMLWrapper* and the LLMs. Especially for the classification of paint defects, the training times were remarkably short, which made it possible to quickly create a selection of prototypes. Although the expected user-friendliness is a strength, it also results in reduced flexibility. The user has less control over specific model parameters and the optimization of these. In addition, the performance of the AutoML-Wrapper is directly dependent on the capabilities of the underlying libraries. In summary, the module offers an efficient and user-friendly way to quickly perform experiments that also result in powerful models.

5.4. Results of the KG-enhanced Approach

Table 4. Overview of the regression experiments (1 & 2) on tabular data. Distribution of train/val/test sets: 204/10/41

	$\text{Wrapper}_{\text{gluon}}$	$\text{Wrapper}_{\text{keras}}$	$\text{Wrapper}_{\text{sklearn}}$	LLM_{Orca}	[19]	KGE_1	KGE_2
Mean-Absolute-Error	8.86	64.68	7.38	53.39	4.58	15.67	0.69
t_{prepare}(min.)	2	2	2	2		1	1
t_{code}(min.)	<1	<1	<1	5		< 1	< 1
t_{compute}(min.)	<1	7	60	18		< 1	25
$work_{\text{code}}$	■□□□□	■□□□□	■□□□□	■■□□□		■□□□□	■□□□□
Prior knowledge	■□□□□	■□□□□	■□□□□	■□□□□		■□□□□	■□□□□
Programming expertise	■□□□□	■□□□□	■□□□□	■□□□□		■□□□□	■□□□□
Initial model	■■■■□	■■■□□	■■■■■	■■■■□		■□□□□	■□□□□

In Table 4, we present an evaluation of the selected dataset for tabular regression using three approaches: the *AutoMLWrapper*, LLM-orchestrated ML code generation, and our proposed knowledge-graph–enhanced method (KGE). Since OpenML is primarily designed for tabular data and contains only limited image-related datasets, this regression task represents the sole case study we report for the KGE approach. The corresponding output is illustrated in Figure 7, which shows the report generated by the AssistML backend when extended with the MLSea knowledge graph. The report lists several candidate

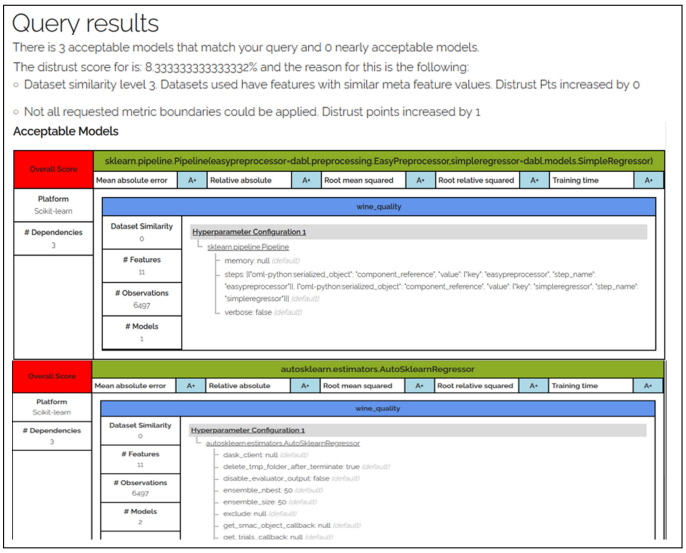

Fig. 7. A report with two model recommendations generated by AssistML [46] for the tabular regression dataset.

models, ordered by decreasing relevance. Notably, the top-ranked recommendation exhibits poor predictive performance, whereas the second-ranked recommendation, an AutoSklearn model, achieves near-perfect alignment with the test data, outperforming even the baseline established in [19]. The superior regression performance of the configuration recommended by AssistML can be primarily attributed to the explicit configuration which departs from the AutoMLWrapper by dramatically increasing the memory limit (102 GB vs. the 4 GB default) and replacing cross-validation with a holdout strategy, thereby enabling more complex regressors to be evaluated within the time budget. Together with constraining the ensemble to the top 50 models, these adjustments substantially broaden the effective search space and account for the observed reduction in regression error. However, due to this configuration, the execution time is much longer.

5.5. Discussion

Applying LLMs for code generation and process understanding in ML represents an innovative and promising approach. The open-source models offer the advantages of free availability, easy access to experimentation without direct costs, and special hardware. However, the open-source models' generated answers were often incomplete or lacked precision for a direct application. Many times the code had errors, was not executable. This is also true for older proprietary models such as GPT3.5 (see this notebook for a comparison between various open-source models and GPT3.5, 4o & 5). For the AutoSklearn model as recommened by the KGE method (see fig. 7), GPT3.5 ommited important model configurations. Especially for more complex queries for model creation and optimization, the answers were cut off prematurely or were so erroneous that considerable effort was required from the user. The results indicated that the models have difficulty understanding

and adapting to specific requirements and contexts. This led to generic or inappropriate code, especially for specialized tasks such as image segmentation. In contrast, *GPT*4 provided a significantly improved experience, generating more precise and relevant code that was better adapted to the problem. The answers were almost complete and pointed to additional aspects that could be used directly as an initial model for further questions to improve the model. For example, the open-source models were unable to utilize the COCO format of the object detection dataset, which was no problem for *GPT*4.

6. Conclusion

In this paper, we explored the integration of AutoML, LLMs, and knowledge graphs to enhance the automation of data science workflows in a data lake environment. While classical AutoML frameworks provide a structured approach to model selection and optimization, they lack flexibility in handling complex, domain-specific tasks. LLMs, on the other hand, offer a more adaptable solution for automating various stages of the ML pipeline but suffer from hallucinations, opaqueness, and difficulty in managing dependencies between tasks.

The evaluation, conducted on real-world industrial datasets, demonstrated that AutoML simplifies pipeline creation but limits user control, while LLM-based approaches require significant manual intervention to refine generated solutions. The findings indicate that neither AutoML nor LLMs alone can fully address the challenges of ML automation, highlighting the need for hybrid approaches.

To address these limitations, we propose a unified approach that integrates knowledge graphs with LLM-based data science agents, leveraging structured knowledge to improve model recommendations. By incorporating *MLSea*, a knowledge graph that aggregates ML datasets, tasks, and models, we introduce a recommendation pipeline that leverages historical ML knowledge for enhanced automation. Initial experiments indicate that incorporating structured machine-learning knowledge into data-science agents enhances the accuracy of automated ML workflows, thereby providing a successful proof of concept that will be subjected to comprehensive evaluation in a subsequent study.

Future work will focus on refining the synergy between these technologies, enhancing the explainability of recommendations, and optimizing multi-agent collaboration for end-to-end ML automation. The combination of the strengths of both frameworks could result in a new ML agent, based on *MLSea* and the derived knowledge, which could propose both an appropriate method for a new dataset and the required hyperparameters. Integrated into a data science agent, such as the *Data Interpreter*, this system has the potential to significantly enhance the efficiency, accuracy and explainability of end-to-end automated ML pipelines.

Acknowledgments. This work has been sponsored by the German Federal Ministry of Education and Research in the funding program "Forschung an Fachhochschulen", project i^2DACH (grant no. 13FH557KX0). We thank Vincent Hermann for his contribution.

References

1. Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., Ye, W., Zhang, Y., Chang, Y., Yu, P.S., Yang, Q., Xie, X.: A survey on evaluation of large language models. ACM Trans. Intell. Syst. Technol. 15(3) (mar 2024)
2. Chen, A., Dohan, D.M., So, D.R.: Evoprompting: Language models for code-level neural architecture search (2023)
3. Chen, T., Liu, Y., Su, H., Chang, Y., Lin, Y., Yeh, J., Hsu, W.H.: Dual-awareness attention for few-shot object detection. CoRR abs/2102.12152 (2021)
4. Dasoulas, I., Yang, D., Dimou, A.: Mlsea: A semantic layer for discoverable machine learning. In: European Semantic Web Conference. pp. 178–198. Springer (2024)
5. Durković, M., Lukovac, P., Hadžić, D., Barać, D., Bogdanović, Z.: Data-driven traffic management: Enhancing road safety through integrated digital twin technology. Computer Science and Information Systems (00), 58–58 (2025)
6. Galluzzo, Y.: A comprehensive review of the data and knowledge graphs approaches in bioinformatics. Computer Science and Information Systems 21(3), 1055–1075 (2024)
7. Grosnit, A., Maraval, A., Doran, J., Paolo, G., Thomas, A., Beevi, R.S.H.N., Gonzalez, J., Khandelwal, K., Iacobacci, I., Benechehab, A., et al.: Large language models orchestrating structured reasoning achieve kaggle grandmaster level. arXiv preprint arXiv:2411.03562 (2024)
8. Gu, K., Shang, R., Jiang, R., Kuang, K., Lin, R.J., Lyu, D., Mao, Y., Pan, Y., Wu, T., Yu, J., et al.: Blade: Benchmarking language model agents for data-driven science. arXiv preprint arXiv:2408.09667 (2024)
9. Guo, S., Deng, C., Wen, Y., Chen, H., Chang, Y., Wang, J.: DS-agent: Automated data science by empowering large language models with case-based reasoning. In: Proceedings of the 41st International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 235, pp. 16813–16848. PMLR (2024)
10. Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N.V., Wiest, O., Zhang, X.: Large language model based multi-agents: A survey of progress and challenges. arXiv preprint arXiv:2402.01680 (2024)
11. Hassan, M.M., , et. al.: ChatGPT as your personal data scientist (2023)
12. Hogan, A., Dong, X.L., Vrandečić, D., Weikum, G.: Large language models, knowledge graphs and search engines: A crossroads for answering users' questions. arXiv preprint arXiv:2501.06699 (2025)
13. Hollmann, N., et al.: Large language models for automated data science: Introducing caafe for context-aware automated feature engineering. Advances in Neural Information Processing Systems 36 (2024)
14. Hong, S., Lin, Y., Liu, B., Liu, B., Wu, B., Zhang, C., Wei, C., Li, D., Chen, J., Zhang, J., et al.: Data interpreter: An llm agent for data science. arXiv preprint arXiv:2402.18679 (2024)
15. Hong, S., Zheng, X., Chen, J., Cheng, Y., Wang, J., Zhang, C., Wang, Z., Yau, S.K.S., Lin, Z., Zhou, L., et al.: Metagpt: Meta programming for multi-agent collaborative framework. arXiv preprint arXiv:2308.00352 (2023)
16. Hoseini, S., Ali, A., Shaker, H., Quix, C.: SEDAR: A semantic data reservoir for heterogeneous datasets. CIKM '23, ACM (2023)
17. Hoseini, S., Burgdorf, A., Paulus, A., Meisen, T., Quix, C., Pomp, A.: Challenges and opportunities of llm-augmented semantic model creation for dataspaces. In: Meroño-Peñuela, A., Corcho, Ó., Groth, P., Simperl, E., Tamma, V., Nuzzolese, A.G., Poveda-Villalón, M., Sabou, M., Presutti, V., Celino, I., Revenko, A., Raad, J., Sartini, B., Lisena, P. (eds.) The Semantic Web: ESWC 2024 Satellite Events - Hersonissos, Crete, Greece, May 26-30, 2024, Proceedings, Part II. Lecture Notes in Computer Science, vol. 15345, pp. 183–200. Springer (2024), https://doi.org/10.1007/978-3-031-78955-7_17

18. Hoseini, S., et al.: Automated defect detection for coatings via height profiles obtained by laser-scanning microscopy. MLWA 10, 100413 (2022)
19. Hoseini, S., et. al.: Coatings intelligence: Data-driven automation for chemistry 4.0. In: 2024 IEEE 7th (ICPS). pp. 1–8 (2024)
20. Hoseini, S., Ibbels, M., Quix, C.: Enhancing machine learning capabilities in data lakes with automl and llms. In: Tekli, J., Gamper, J., Chbeir, R., Manolopoulos, Y. (eds.) Advances in Databases and Information Systems - 28th European Conference, ADBIS 2024, Bayonne, France, August 28-31, 2024, Proceedings. Lecture Notes in Computer Science, vol. 14918, pp. 184–198. Springer (2024), https://doi.org/10.1007/978-3-031-70626-4_13
21. Hoseini, S., Theissen-Lipp, J., Quix, C.: A survey on semantic data management as intersection of ontology-based data access, semantic modeling and data lakes. Journal of Web Semantics 81, 100819 (2024)
22. Hu, L., Liu, Z., Zhao, Z., Hou, L., Nie, L., Li, J.: A survey of knowledge enhanced pre-trained language models. IEEE Transactions on Knowledge and Data Engineering (2023)
23. Huang, Q., Vora, J., Liang, P., Leskovec, J.: Benchmarking large language models as ai research agents (2023)
24. Iglesias, E., Jozashoori, S., Vidal, M.E.: Scaling up knowledge graph creation to large and heterogeneous data sources. Journal of Web Semantics 75 (2023)
25. Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y.J., Madotto, A., Fung, P.: Survey of hallucination in natural language generation. ACM Comput. Surv. 55(12) (mar 2023)
26. Karmaker, S.K., et al.: Automl to date and beyond: Challenges and opportunities. ACM Comput. Surv. 54(8) (oct 2021)
27. Khan, Y., Zimmermann, A., Jha, A., Gadepally, V., d'Aquin, M., Sahay, R.: One size does not fit all: querying web polystores. Ieee Access 7, 9598–9617 (2019)
28. Kreuzberger, D., Kühl, N., Hirschl, S.: Machine learning operations (mlops): Overview, definition, and architecture. IEEE Access 11, 31866–31879 (2022)
29. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems 25 (2012)
30. Li, D., Tan, Z., Liu, H.: Exploring large language models for feature selection: A data-centric perspective. ACM SIGKDD Explorations Newsletter 26(2), 44–53 (2025)
31. Li, X., Döhmen, T.: Towards efficient data wrangling with llms using code generation. In: Proceedings of the Eighth Workshop on Data Management for End-to-End Machine Learning. pp. 62–66 (2024)
32. Liu, S., Gao, C., Li, Y.: Large language model agent for hyper-parameter optimization. arXiv preprint arXiv:2402.01881 (2024)
33. Malberg, S., Mosca, E., Groh, G.: Felix: Automatic and interpretable feature engineering using llms. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 230–246. Springer (2024)
34. Peng, B., Zhu, Y., Liu, Y., Bo, X., Shi, H., Hong, C., Zhang, Y., Tang, S.: Graph retrieval-augmented generation: A survey. arXiv preprint arXiv:2408.08921 (2024)
35. Shen, Y., Song, K., Tan, X., Li, D., Lu, W., Zhuang, Y.: Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. Advances in Neural Information Processing Systems 36 (2024)
36. Singh, S.K., Cha, J., Kim, T.W., Park, J.H.: Machine learning based distributed big data analysis framework for next generation web in iot. Computer Science and Information Systems 18(2), 597–618 (2021)
37. Sun, M., Han, R., Jiang, B., Qi, H., Sun, D., Yuan, Y., Huang, J.: A survey on large language model-based agents for statistics and data science. arXiv preprint arXiv:2412.14222 (2024)
38. Sun, Y., Yan, B., Shao, Z.: Ai large models bring great opportunities to reusable design of cad software. Computer Science and Information Systems (00), 46–46 (2024)
39. Sutskever, I.: Sequence to sequence learning with neural networks: What a decade. https://www.youtube.com/watch?v=WQQdd6qGxNs, access on 19.02.2025 (2024), neurIPS 2024

40. Tornede, A., et al.: Automl in the age of large language models: Current challenges, future opportunities and risks. arXiv preprint arXiv:2306.08107 (2023)
41. Trirat, P., Jeong, W., Hwang, S.J.: Automl-agent: A multi-agent llm framework for full-pipeline automl. arXiv preprint arXiv:2410.02958 (2024)
42. Truong, A., Walters, A., Goodsitt, J., Hines, K., Bruss, C.B., Farivar, R.: Towards automated machine learning: Evaluation and comparison of automl approaches and tools. In: 2019 IEEE 31st international conference on tools with artificial intelligence (ICTAI). pp. 1471–1479. IEEE (2019)
43. Wei, L., He, Z., Zhao, H., Yao, Q.: Unleashing the power of graph learning through llm-based autonomous agents (2023)
44. Yang, R., Liu, H., Zeng, Q., Ke, Y.H., Li, W., Cheng, L., Chen, Q., Caverlee, J., Matsuo, Y., Li, I.: Kg-rank: Enhancing large language models for medical qa with knowledge graphs and ranking techniques. arXiv preprint arXiv:2403.05881 (2024)
45. Yang, Z., , et. al.: Autommlab: Automatically generating deployable models from language instructions for computer vision tasks (2024)
46. Zacarias, A.G.V., Reimann, P., Weber, C., Mitschang, B.: Assistml: an approach to manage, recommend and reuse ML solutions. Int. J. Data Sci. Anal. 16(4), 455–479 (2023), https://doi.org/10.1007/s41060-023-00417-5
47. Zaharia, M., et al.: Accelerating the machine learning lifecycle with mlflow. IEEE Data Eng. Bull. 41(4), 39–45 (2018)
48. Zhang, G., et al.: Deep learning-based automated characterization of crosscut tests for coatings via image segmentation. JCTR 19, 671–683 (2021)
49. Zhang, K.: Using deep learning to automatic inspection system of printed circuit board in manufacturing industry under the internet of things. Computer Science and Information Systems 20(2), 723–741 (2023)
50. Zhang, L., Zhang, Y., Ren, K., Li, D., Yang, Y.: Mlcopilot: Unleashing the power of large language models in solving machine learning tasks. arXiv preprint arXiv:2304.14979 (2023)
51. Zhang, S., Gong, C., Wu, L., Liu, X., Zhou, M.: Automl-gpt: Automatic machine learning with gpt. arXiv preprint arXiv:2305.02499 (2023)

Sayed Hoseini is a researcher at the Hochschule Niederrhein University of Applied Sciences (HSNR) pursuing a PhD degree from RWTH Aachen University, Germany. His research focuses on data architectures specifically data lakes, data integration, and data management for machine learning. He has served as a program committee member of database conferences such as CIKM, ADBIS, and as journal reviewer for MLWA.

Maximilian Ibbels is a software engineer who graduated in 2024 from HSNR with a Master of Science in Computer Science. His master's thesis focused on extending the SEDAR data lake system with automated machine learning capabilities. He developed and evaluated the AutoMLWrapper and implemented LLM-based code generation to support automated ML model creation.

Maximilian Knoll is a software engineer who graduated in 2025 from HSNR with a Master of Science in Computer Science. His master's thesis focused on developing an LLM-based Data Science agent for automated machine learning model generation. He integrated the MLSea knowledge graph with the AssistML framework to create a semantics-aware agent capable of supporting and automating the ML model development process.

Christoph Quix is professor for Information Systems and Data Science at HSNR. He is also lead scientist for smart data management and analytics in the Fraunhofer Institute for Applied Information Technology FIT. His research focuses on data integration, data science, management of large, heterogeneous data sets, and metadata management in different application areas, e.g, chemistry 4.0 and industry 4.0. He has more than 100 publications in international journals and conferences.

Received: March 20, 2025; Accepted: November 15, 2025.