From Natural Language to Executable ETL Flows: The IBM DataStage Assistant

Nitin Gupta[1*], Thomas Gschwind[1*], Shramona Chakraborty[1*], Sameep Mehta[1], Tristan Tyler[2], Shreya Sisodia[2], Ben Clermont[2]

[1]IBM Research,
[2]IBM Software

ngupta47@in.ibm.com, thg@zurich.ibm.com, shramona.chakraborty1@ibm.com, sameepmehta@in.ibm.com, tristan.tyler@ibm.com, shreya.sisodia@ibm.com, ben.clermont@ibm.com

Abstract

Modern ETL (Extract, Transform, Load) tools offer graphical, no-code interfaces for workflow creation but still require users to manually identify transformation functions and configure their properties, which is time-consuming and demands prior expertise. We present the research and engineering foundations of the IBM DataStage Assistant, a deployed capability that generates complete multi-stage ETL flows directly from natural language (NL) descriptions. Our framework infers transformation functions, their properties, and transformer expressions, enabling novices to discover relevant functions and allowing experts to bypass manual configuration. The proposed framework achieves a prediction accuracy of 96.4% for flow predictions, 87.0% for properties, and 83.6% for transformer expressions. We also show a document exploration module that uses retrieval-augmented generation (RAG) over product documentation to answer tool-specific questions in NL. Implemented in IBM DataStage, this approach supports iterative, in-environment workflow design and reduces context switching. In initial studies, it achieves up to 90% time savings for novices and 50% for experts.

Introduction

Most organizations manage massive quantities of data integral to their business operations, such as financial transactions or client records. To make this diverse data usable for analytics and decision-making, organizations rely on Extract, Transform, and Load (ETL) processes—three distinct but interdependent steps that integrate data from various sources into a cohesive whole (Vassiliadis 2009). This integration not only leverages technology and business methods but also produces accurate, consolidated datasets ready for business use, thereby enabling a DataOps-driven data flow.

In the early stages of ETL adoption, organizations typically hand-coded ETL processes, creating bespoke software solutions. Managing such software was time-consuming and costly, prompting the emergence of ETL tools to simplify development. (Vassiliadis et al. 2001) identified challenges in these tools, including pricing, complexity, ease of use, and maintenance. To address these concerns, alternative approaches were proposed, such as ontology-based methodologies to improve efficiency and semantics in data warehouse development (Jiang, Cai, and Xu 2010) and GUI-based ETL creation for more streamlined data imports into active data warehouses (Reddy and Jena 2010). These tools enabled no-code graphical workflows but still required users to (a) know the available transformation functions and their properties, and (b) manually navigate to each property to set the appropriate value.

To further lower the skill and effort barriers, recent research and commercial tools have begun exploring natural language (NL) interfaces for data manipulation and workflow creation. While prior work in this area has focused on SQL query generation (Zhong, Xiong, and Socher 2017; Wang et al. 2021a; Scholak, Schucher, and Bahdanau 2021) or generic workflow automation (Brachman et al. 2022; Dhamdhere et al. 2017), no existing system supported orchestrating complete multi-stage ETL pipelines directly from natural language until now.

In this work, we present the IBM DataStage Assistant, a released capability embedded directly within IBM DataStage that allows users to specify one or more ETL operations in natural language. From these utterances, the system automatically infers the appropriate connectors, transformation operators, and their associated properties, producing fully executable ETL flows. Novice users can simply verify the generated predictions without needing to understand the underlying transformation details, while expert users benefit from pre-filled workflows that eliminate much of the manual property configuration effort.

Within this broader functionality, an important advanced capability is the automation of the Transformer stage, a powerful yet complex operator in DataStage that enables custom row-level transformation logic. Traditionally, authoring Transformer logic requires detailed knowledge of stage-specific syntax, available functions, and variable handling. Our system reduces this complexity by allowing users to describe their desired transformation in natural language, then automatically generating the corresponding Transformer stage code using a curated library of function specifications and operator behaviors. This not only accelerates development but also lowers the barrier for users unfamiliar with the Transformer's details.

To further reduce the time and effort in building flows, we

*These authors contributed equally to this work.

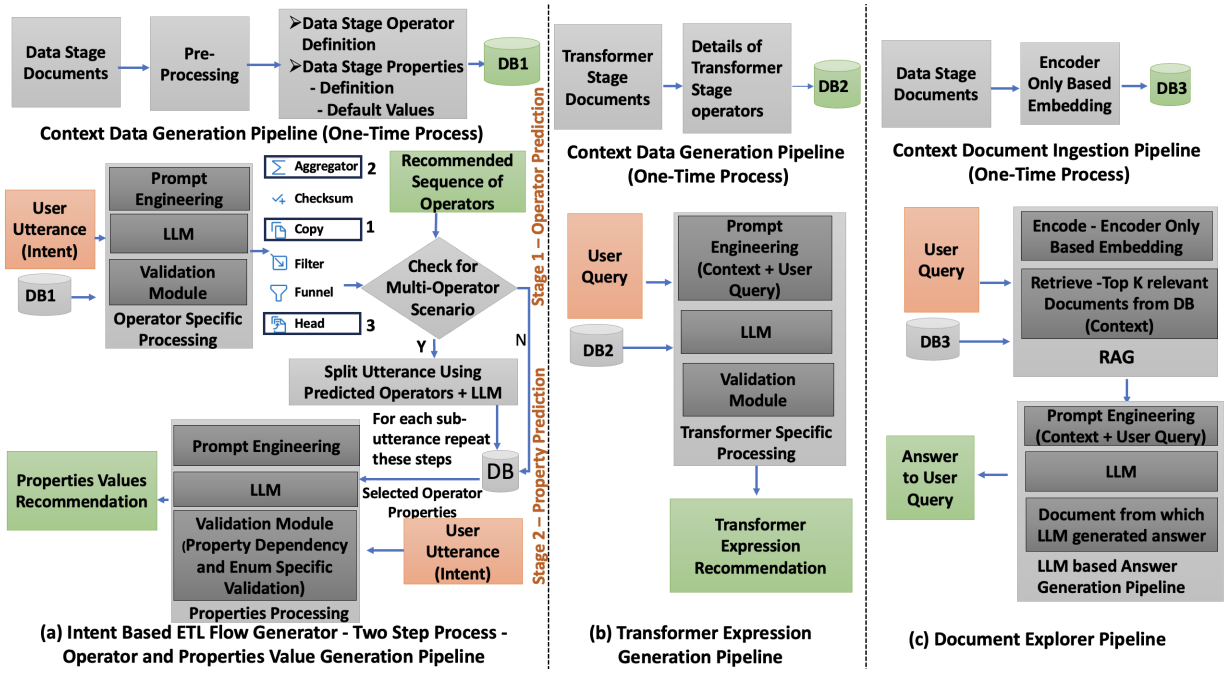

Figure 1: Proposed Architecture for ETL Flow Creation, Transformer Expression Generation and Document Explorer

integrate a Document Explorer that enables natural language queries over product documentation, including DataStage stages, connectors, transformation functions, property descriptions, installation steps, and usage guidelines. While document exploration itself is a well-studied area, we incorporate it as a complementary capability to accelerate the ETL design process, particularly by reducing context switching between design and reference material. This component uses a RAG+LLM approach to retrieve and answer documentation-related queries.

The three capabilities: Flow Generation, Transformer Stage Automation, and Document Exploration, work together to improve efficiency. Flow generation accelerates design, transformer automation reduces the effort of authoring complex transformation logic, and document exploration resolves uncertainties in real time. Together, they enable an iterative workflow where users refine designs and clarify questions without leaving the DataStage environment. Our implementation supports the full range of DataStage connectors, stages, and transformation functions*, ensuring applicability in enterprise workflows, and is designed to scale with production workloads while meeting governance and audit requirements.

The IBM DataStage Assistant is now released and available to all DataStage users. In internal testing with several participants (experts and novices), experts reported approximately 50% time savings by avoiding manual lookups and benefiting from 80% of property prediction handled automatically. Novices highlighted the combined benefit of flow assist and document exploration, achieving up to a 90% reduction in ETL creation time while easily finding answers to tool-specific questions. In addition, automation of Transformer stage expressions benefited both experts and novices, with participants reporting savings of nearly 80–90% of the time otherwise required to manually write or configure complex transformation logic. These results demonstrate measurable productivity gains and confirm the Assistant's value as an integrated, enterprise-grade natural language interface for ETL design.

*https://dataplatform.cloud.ibm.com/docs/content/dstage/dsnav/topics/datastage-create-flow.html

Related Work

Several systems have explored natural language (NL) interfaces for data manipulation and workflow creation, motivated by the limitations of GUI-based ETL tools that still require detailed knowledge of available transformations and manual property configuration. None of the popular ETL tools (Datacamp Team 2023) support end-to-end flow description in natural language, and existing efforts either target single-stage data access or generic automation rather than orchestrating complete multi-stage ETL pipelines.

Analyza (Dhamdhere et al. 2017) converts NL input into SQL queries using a parser, annotator, and table identifier. To handle ambiguity in complex queries, it initially focuses on simple ones and iteratively refines them, conceptually similar to our staged ETL design. However, its scope is limited to query formulation, without transformation logic or flow generation, and it provides no quantitative evaluation.

GOFA (Brachman et al. 2022), similar in spirit to Analyza, generates application integration workflows from NL using a Knowledge Graph, removing the need for an annotator. While it can produce a sequence of API calls from a single utterance, it is aimed at enterprise application integra-

tion, not structured ETL over heterogeneous data sources.

Commercial offerings such as Zapier's AI-based Zap Builder (Zapier 2023) and Microsoft Power Automate (Microsoft n.d.) claim GPT-driven flow creation, but their proprietary nature and lack of peer-reviewed evaluation make them unsuitable for reproducible benchmarking.

Text-to-SQL systems like Seq2SQL (Zhong, Xiong, and Socher 2017), RAT-SQL (Wang et al. 2021a), and PICARD (Scholak, Schucher, and Bahdanau 2021) achieve high accuracy in NL→SQL translation, yet are designed for database querying, not multi-stage ETL orchestration. Likewise, NL-to-code models such as Codex (Chen et al. 2021) and CodeT5 (Wang et al. 2021b) can generate scripts from descriptions but lack built-in abstractions for ETL stages, dependency management, and dataflow validation.

In contrast, our work focuses specifically on generating complete ETL execution flows from natural language descriptions, encompassing extraction, transformation, and loading stages, with integrated document-driven exploration to reduce pipeline design time and minimize user cognitive load. This approach not only streamlines multi-stage pipeline creation for both novice and expert users but is also delivered as a generally available feature within the IBM DataStage product, ensuring scalability, governance, and seamless integration into enterprise environments.

Architecture

The overall system architecture is shown in Figure 1 and consists of three main components: the *Intent-Based ETL Flow Generator*, the *Transformer Expression Generator*, and the *Document Explorer*. These components work together to accelerate ETL design and reduce user effort. The Flow Generator leverages large language models (LLMs) to map natural language utterances to ETL operators and their properties, the Transformer Expression Generator automates the creation of complex transformation logic from natural language, and the Document Explorer uses retrieval-augmented generation (RAG) to answer tool-specific questions in real time.

Intent-Based ETL Flow Generator

The Flow Generator (Figure 1a) translates user utterances into ETL workflows. Before runtime, we perform a one-time analysis of the ETL tool's documentation to identify supported stages and their properties, including descriptions, mandatory conditions, default values, and types. This information forms a *context database* used for prediction and validation (Figure 2).

At runtime, we apply in-context learning with stage descriptions, few-shot examples, and the user utterance to infer both the stage names and their associated properties. Since LLMs have limited context capacity and may produce inaccurate outputs, we adopt a two-step process: first predict the stage(s), then predict their properties.

Formally, let $\mathcal{S} = \{s_1, s_2, \ldots, s_m\}$ denote the set of all ETL stages supported by the tool, where each stage s_i has an associated set of properties $\mathcal{P}_i = \{p_{i1}, p_{i2}, \ldots, p_{in_i}\}$. Given a natural language utterance u, the goal of the *Intent-Based ETL Flow Generator* is to predict

$$f_{\text{stage}} : u \mapsto \hat{\mathcal{S}} \subseteq \mathcal{S}$$

and, for each predicted stage $\hat{s}_i \in \hat{\mathcal{S}}$, a function

$$f_{\text{prop}} : (u, \hat{s}_i) \mapsto \hat{\mathcal{P}}_i$$

where $\hat{\mathcal{P}}_i$ is a valid assignment of property values satisfying all constraints defined in the context database.

Operator Prediction After experimenting with multiple designs, we found that the most effective prompt includes operator names with their descriptions, along with few-shot examples covering both single and multiple operator predictions. To handle cases where no suitable match exists, we introduced an artificial operator, "unknown," which the LLM can select when appropriate. Using this prompt, the LLM predicts one or more operators for a given user utterance. To mitigate hallucinated outputs, we applied two validation strategies: (i) restricting predictions to operators defined in our context database (including "unknown"), and (ii) prompting the user for additional details when no valid match is identified. The validated operator prediction is then passed on to the subsequent stage of the workflow.

Operator Prompt

Instruction: Understand the context, given in the form of operator and their descriptions, assign the correct operators to the Utterance. Make sure to choose from the list of the operators provided.

Context:

"{operator1 name}": {operator1 definition}
...
"{operatorN name}": {operatorN definition}

Training Examples:

Example 1:
User Utterance: {task1}
Operator: "{operator1 name}"
...
Example N:
User Utterance: {taskN}
Operator: "{operatorN name}"

Test Example:
User Utterance: {task}
Operator:

Property Prediction To predict the properties associated for the operator(s) predicted in the previous step, we choose few-shot samples for ICL from the same class. Initially, we compared predicting one property at a time versus all at once and observed that the latter gives the better performance. One possible reason could be that certain properties within the operator are interdependent, and predicting them together provides more contextual information.

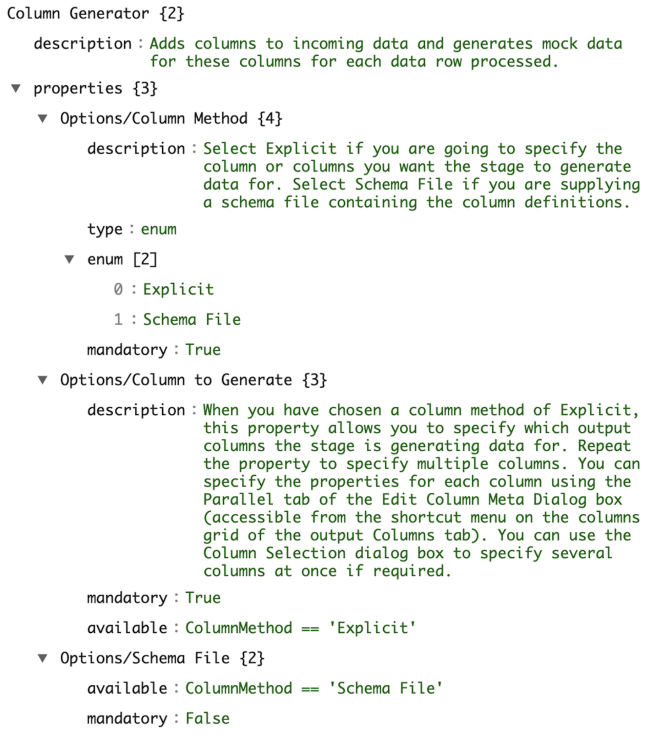

Figure 2: Extracted Context Data

Initially, we used a prompt containing the properties, their descriptions, and a single sample yields optimal results. This approach was particularly beneficial in cold-start scenarios with access to only a limited set of samples. As we gathered more data points, we increased the number of samples to improve the algorithm's accuracy through few-shot learning. When a larger number of samples per operator are available, dynamic sample selection can be used for optimizing the samples to include in the prompt.

For each predicted operator, we send the below prompt to the LLM to predict the properties. However, LLMs can confuse which property in the user's utterance corresponds to which operator, especially when the same operator appears multiple times. To mitigate this, we ask the LLM to break the user's utterance into smaller segments corresponding to the predicted operators and only pass these segments with their respective operators to the LLM which increases prediction accuracy by about 10%.

We also implement a validation strategy to manage incorrect properties generated by the LLM. The validation process starts by examining the properties associated with the selected task. We assess the relationships between properties within the same operator. For example, within the *Column Generator* operator, the *Options/Column to Generate* property depends on the *Options/Column Method* property value being *Explicit*. We then evaluate each property's nature, determining whether it is an enumeration, names a database table or column, or is present in the user utterance. For instance, the *Options/Column Method* property is an enumeration limited to *Explicit* or *Schema File* (see Figure 2).

Properties Prompt

Instruction: Use below API parameters definition and find all the API parameter values from the Utterance for the {op} function.

API Parameters Definitions:
{op property name 1} = {op property 1 definition}
–
{op property name M} = {op property M definition}

Training Example:
User Utterance - {task}
API - {task operator}

API Parameters Values:
{op property name 1} = {op property 1 extracted value}
–
{op property name M} = {op property M extracted value}

Test Example:
User Utterance - {utterance}
API - {op}

API Parameters Values:

Transformer Expression Generator

This component (Figure 1b) handles the *Transformer* stage, where users specify custom transformation logic in the form of expressions. The central challenge is designing few-shot prompts that encode the grammar of the supported function library, enabling the LLM to generate syntactically valid and semantically meaningful expressions directly from natural language descriptions. We use below prompt to generate transformer expression, which demonstrates how function signatures and usage examples are provided to guide the model toward producing valid outputs.

At runtime, the system generates a candidate expression, which is then passed through a validation layer to ensure correctness. The validator checks that the expression only uses functions from the supported library, that the number of arguments matches the expected function arity, and that argument types are consistent with the function signatures.

Formally, let $\mathcal{F} = \{f_1, \ldots, f_r\}$ denote the set of supported functions, where each f_j has a defined arity $\text{arity}(f_j)$ and argument types $\text{type}(f_j)$. Given a NL description u, the goal of the *Transformer Expression Generator* is to produce: $h(u) \rightarrow \hat{e}$, such that $\hat{e}$ is an expression over $\mathcal{F}$ and, for every occurrence of f_j in $\hat{e}$, the number of arguments equals $\text{arity}(f_j)$ and the argument types match $\text{type}(f_j)$.

Document Explorer

This component (Figure 1c) enables users to interact with the system to obtain answers related to various aspects of ETL tools. It comprises two parts. First, we perform a one-time setup by analyzing documents and converting them into text format to create a document database. We break larger documents into smaller chunks to comply with LLM context limits. Second, the system allows users to query information

about the ETL tool. It scans the documents for relevant information and generates responses. Since including all documents in context is not feasible, we employ a retrieval-augmented generation (RAG) approach, which effectively selects relevant documents for a given query by retrieving the top-k documents using BERT (Devlin et al. 2019).

Transformer Expression Generator

You are a helpful AI assistant. Generate a simple function code from the given utterance using the functions exclusively from the provided function list, the syntax is based on IBM DataStage's own proprietary expression language. If you cannot come up with a function, notify the user that you don't know the code. Do not return anything other than valid generated code. Enclose your final answer in `<code>` and `</code>` tags. Do not provide any further explanation.

{Language_syntax_description}

Function List: [...]
Function Details:
{
'Subheading': {Function Heading},
'Title': {Function Title},
'Description': {Function Description},
'Input': {Function Input},
'Output': {Function Output}
}
Fewshot examples (columns may be provided, but not all necessarily used):
Utterance: {Utterance1}
SourceColumn: {SourceColumn Names}

Code: {code1}
—-
Utterance: {UtteranceN}
SourceColumn: {SourceColumn Names}

Code: {codeN}

Understand the utterance properly and do not create or guess any new function which are not present in the function list. Apply column based on utterance and column type. Generate only the code.
Utterance: {Test Utterance}
SourceColumn: {SourceColumn Names}

Code:

We also retrieve the name of the document from which the result is derived, aiming to uphold transparency between the output and the user. To do so, we identify the longest sub-string in the generated answer that matches the content of the retrieved document, by maximizing word matches.

Formally, let $\mathcal{D} = \{d_1, \ldots, d_k\}$ denote the set of text chunks indexed in the document database. Given a user query q, the goal of the *Document Explorer* is to retrieve the most relevant subset $\hat{\mathcal{D}} \subseteq \mathcal{D}$ and generate an answer

$$g(q, \hat{\mathcal{D}}) \rightarrow a$$

such that a is semantically aligned with q and grounded in the retrieved documentation.

Experiments

In order to assess the efficiency and functionality of the system, we conducted a comprehensive evaluation using a specially curated dataset. Since there is no open-source dataset available for such evaluation, we constructed the required data from *IBM DataStage*, whose documentation is publicly accessible. The same process could be replicated for any other such ETL tool.

As part of the one-time setup for flow generation module, we extracted 142 *DataStage* stages (90 of them being data source connectors) along with their descriptions and properties from the official documentation. Stages, also referred to as data sources, operators, or tasks, form the fundamental building blocks of an ETL workflow. Each stage contains between 1 and 111 properties, with an average of 27.6. For every property, we store its description, type, default value, and availability conditions in a structured format suitable for prompt generation and output validation.

To evaluate our system, we created a ground-truth dataset that includes $1,010$ natural language flow descriptions for stage prediction. Each of these utterances was thoughtfully crafted to represent various scenarios and user inputs, and annotated with ground-truth operator names and corresponding property values. Of these, 700 correspond to single-stage workflows, while 310 involve multiple stages combined into a single flow. From this set, 308 flows were further annotated with a total of $1,410$ properties. In the operator prediction prompt, we included 142 few-shot examples along with task instructions, stage names with one-line descriptions, and the user's utterance. These examples help compensate for the LLM's limited pretrained knowledge about individual stages by illustrating how stages are combined in real ETL tasks. On average, each stage appears in approximately two examples. We evaluate the performance of both operator and property predictions with respect to their accuracy.

In addition, we curated 483 test utterances specifically for evaluating the *Transformer Expression Generator*, spanning 21 categories, including complex and compositional cases. The *Transformer* stage in *DataStage* supports 261 operators for expression-based transformations, and we targeted these in our evaluation. To aid the model, we constructed 445 few-shot examples that capture both syntactic and semantic variations of expressions, illustrating the mapping from natural language instructions to valid function calls. The evaluation checks both the correctness of the generated expressions and their validity against the function grammar, including correct function names, arity, and parameter types.

For the document explorer, we compiled 458 query–answer pairs from 319 unique documents, each derived from a thorough review of all available documentation. These queries were designed to cover a diverse

Model	Operator Prediction Accuracy [%]			Property Accuracy [%]	Transformer Expression [%]
	Total	1-op	n-op		
granite-3.1-8b-instruct	88.0	92.6	77.7	81.8	83.6
llama-3.2-3b-instruct	71.1	92.3	23.2	72.1	12.8
llama-3.1-8b-instruct	91.6	96.0	81.6	81.8	58.8
llama-3.3-70b-instruct	96.4	98.1	92.6	87.0	81.8
llama-4-maverick-17b-128e-instruct-fp8	95.8	97.7	91.6	67.3	77.2
gpt-oss-20b	75.4	74.9	76.8	49.8	77.4
gpt-oss-120b	94.5	94.6	94.2	78.9	79.7

Table 1: Combined Model Performance: Operator Prediction Accuracy ("1-op" refers to predictions involving a single stage; "n-op" refers to predictions involving two or more stages), Property Prediction Accuracy, and Transformer Expression Accuracy

range of information needs, enabling us to comprehensively assess the system's proficiency in navigating, retrieving, and synthesizing content from technical documents.

Results and Discussion

We evaluated our system on 7 recent and widely known LLMs: *granite-3.1-8b-instruct* (IBM 2025), *llama-3.1-8b-instruct* (Meta 2024), *llama-3.2-3b-instruct* (Meta 2024), *llama-3.3-70b-instruct* (Meta 2024), *llama-4-maverick-17b-128e-instruct-fp8* (Meta 2025), *gpt-oss-120b*, and *gpt-oss-20b* (OpenAI et al. 2025). All experiments were conducted on the IBM WatsonX AI platform, and for all LLMs we used greedy decoding. For the *gpt-oss* variants, which natively support the Harmony response format, we adapted our prompt templates slightly to align with this format while keeping the evaluation protocol consistent across all models.

Intent-Based ETL Flow Generator

Table 1 reports the results for the individual modules that make up the flow generation. Generally, we can observe that larger models within a given family give better results with the exception of Llama 4 model which given its size falls behind expectations. A similar observation has been made by others (Nuenki 2025; ForsookComparison 2025).

Among all models, *llama-3.3-70b-instruct* achieves the highest overall operator accuracy (96.4%), with strong performance on both single-operator (98.1%) and multi-operator (92.6%) cases. *llama-4-maverick-17b-128e-instruct-fp8* follows closely (95.8%), while *gpt-oss-120b* also performs competitively (94.5%). At the mid-scale, *granite-3.1-8b-instruct* achieves a solid balance (88.0% for operators, 81.8%) for properties, making it an efficient choice relative to model size.

For property prediction, *llama-3.3-70b-instruct* again performs best with 87.0%, followed by the *granite-3.1-8b-instruct* and *llama-3.1-8b-instruct* models (both 81.8%). While larger models generally perform better, mid-sized models are not far behind and can offer competitive trade-offs for practical deployment.

Transformer Expression Generator

Table 1 also presents the prediction accuracy of different models for generating Transformer stage expressions. Among smaller models, *granite-3.1-8b-instruct* achieves the best performance (83.6%), substantially outperforming other model of comparable and larger sizes. For larger models, *llama-3.3-70b-instruct* demonstrates strong accuracy (81.8%), closely matched by *gpt-oss-120b* (79.7%) and *gpt-oss-20b* (77.4%), with *llama-4-mvk-17b-128e* also performing competitively (77.2%). These results highlight that expression generation benefits not only from scaling up model size but also from model family characteristics, with the *granite-3.1-8b-instruct* offering an attractive balance between accuracy and computational efficiency.

Category	#Total	#Correct	Accuracy[%]
Mathematical	52	50	96.2
Logical	13	11	84.6
IF THEN ELSE	33	27	81.8
Type Conversion	57	42	73.7
String Nested	11	5	45.5
Nested	3	3	100.0
Utility	24	18	75.0
Outcome-Oriented	31	25	80.7
Loop Variables	9	8	88.9
Raw	1	1	100.0
Null Handling	12	11	91.7
Date and Time	55	46	83.6
Number	20	20	100.0
String	90	77	85.6
Macros	14	14	100.0
Key Break Detection	10	9	90.0
System Variables	7	6	85.7
Stage Variables	15	14	93.3
Date and Time Nested	14	5	35.7
Vector	10	10	100.0
Conversion	2	2	100.0
Summary	483	404	83.6

Table 2: Category-wise Transformer Expression Accuracy statistics for granite-3.1-8b-instruct model.

Table 2 further breaks down the accuracy of *granite-3.1-8b-instruct* across categories. The model shows strong results in core transformations such as *Mathematical* (96.2%), *Stage Variables* (93.3%), and *Null Handling* (91.7%), while achieving perfect accuracy in categories like *Macros*, *Number*, and *Vector*. Performance is weaker for more complex nested constructs, e.g., *String Nested* (45.5%) and *Date and Time Nested* (35.7%), highlighting areas for improvement. Overall, the model attains 83.6% average accuracy across 483 cases, showing strong robustness.

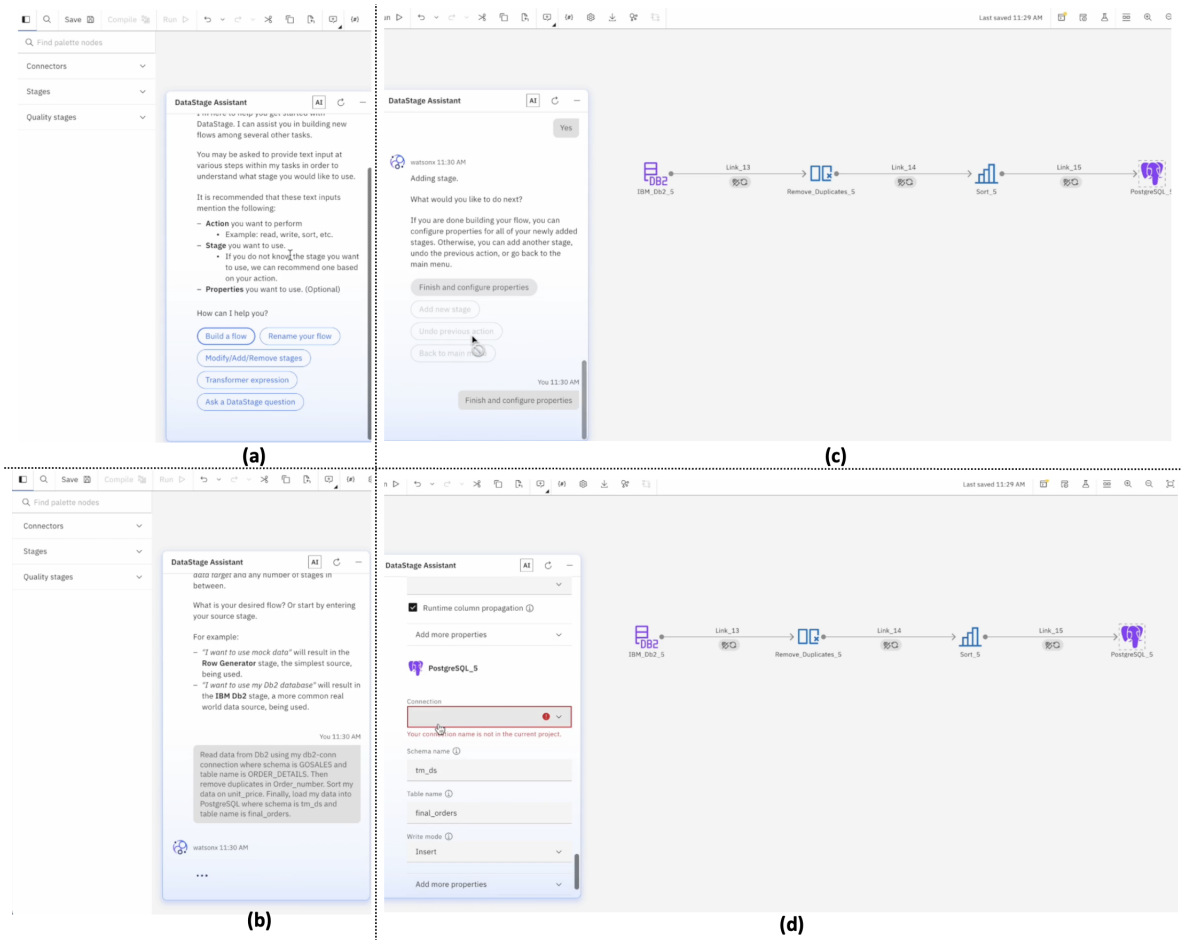

Figure 3: Screenshots of the assistant integrated in IBM DataStage: (a) interaction options, (b) user utterance via 'Build a flow', (c) auto-generated flow with option to configure properties, and (d) system-generated properties (one operator shown), with manual additions possible where needed.

Document Exploration

Although included mainly for completeness, the document exploration module also demonstrated solid performance in retrieving and leveraging relevant content from technical documentation. We measured two aspects: (i) Relevant Document in Prompt (RD_Prompt), which captures whether the retrieved content fit entirely within the LLM prompt context, and (ii) Answer Generation Performance, evaluated using BLEU and ROUGE scores. We experimented with several embeddings, and the *multi-qa-mpnet-base-dot-v1* embedding consistently yielded the highest retrieval relevance, achieving an RD_Prompt value of approximately 77%. Most evaluated LLMs achieved BLEU and ROUGE scores in the 40–60% range, confirming the system's ability to provide accurate answers when relevant content appeared in the top-k retrieved results.

User Study

We conducted a user study involving several participants, divided into two categories: (a) expert users familiar with ETL operators, and (b) novice users with limited knowledge of these operators. Each participant designed an ETL workflow for several use cases in two rounds: first without, and then with the natural language-based feature. Expert users reported significant time savings (approximately 50%) as they no longer needed to refer to operator lists and definitions, along with reduced manual effort and errors, with 80% of property prediction handled automatically by the tool compared to manually adding properties. In addition, automation of transformer stage expressions benefited both experts and novices, with participants reporting savings of nearly 80–90% of the time otherwise required to manually write or configure complex transformation logic. Novice users highlighted the usefulness of both the flow assist and document explorer features for rapid flow creation and answering tool-specific questions, resulting in a 90% reduction in the time required for ETL flow creation.

Deployment and Impact

From Research to Product Deployment

This project began as a research prototype and has evolved through several development cycles of development and re-

finement. Following extensive evaluation and internal validation, the system was integrated into the *IBM DataStage* product and is now generally available (GA) for production use on DataStage SaaS[†] on IBM Cloud Pak® for Data within the Dallas, Frankfurt, and Sydney data centers. The assistant is embedded directly within the product interface, allowing the target persona to generate ETL workflows and query documentation seamlessly. Currently, it is globally accessible to more than 100 DataStage users.

This journey from research concept to enterprise-grade deployment required support for a wide range of operators and stages, robust handling of edge cases, and engineering for reliability and scalability. It helps in onboarding the target persona more easily and enabling experienced developers to accelerate ETL design. This work has gained traction among customers as an incentive for modernization.

Target Personas

The DataStage Assistant targets junior and new DataStage data engineers who possess foundational ETL knowledge but limited familiarity with the DataStage Canvas. This persona is responsible to build and maintain data pipelines but often face a learning curve when navigating the DataStage environment.

User Interaction and Experience

As shown in Figure 3(a), the DataStage Assistant provides a conversational and intuitive interface embedded within the DataStage Canvas, enabling the target persona to design and understand data flows through natural language interaction. As shown in Figure 3(b), the persona can enter instructions such as "Join customer and sales data on customer_id" or "Explain what this transformer stage does," and the DataStage Assistant automatically constructs or annotates the corresponding flows as shown in Figure 3(c) and 3(d) while invoking relevant DataStage interfaces for any required manual configurations like column mappings or key selections. This integration ensures a seamless transition between guided conversation and graphical design. Similarly, the persona can generate the DataStage transformation expression and also interact to explore DataStage documents using the interface shown in Figure 3(a). The current release employs the *IBM granite-3.1-8b-instruct* model as the default backbone, balancing accuracy and efficiency, while still allowing users to choose alternative models available on *watsonx.ai*. Evaluation results further indicate that most models achieve comparable performance, reinforcing the robustness and portability of the overall framework.

Deployment Environment and Platform

The DataStage Assistant backend is deployed as a containerized FastAPI service running on RHEL within existing IBM Cloud/MCSP Kubernetes clusters, colocated with DataStage to ensure enterprise data residency requirements. Each deployment uses Gunicorn with Uvicorn workers (8 per pod) and exposes an asynchronous REST API exclusively to the DataStage Canvas frontend. All executed flow changes require explicit user confirmation within the UI; the assistant does not directly interact with customer data sources or execution runtime. LLM inference is performed through watsonx.ai models, currently rate-limited to 8 requests per second per environment. A lightweight Milvus vector database stores embeddings of documentation and system knowledge for contextual retrieval. The feature is opt-in at the DataStage project level, and additional pod replicas can be enabled to scale with usage demand. The deployment architecture is designed for secure, responsive, and incrementally scalable integration into the DataStage product.

Observed Impact

Preliminary adoption indicates high engagement among target personas, particularly in onboarding scenarios. Users report faster time-to-first-successful-flow creation and improved understanding of complex pipeline logic. The integration of LLM-driven guidance within the DataStage Canvas demonstrates tangible improvements in usability, learning efficiency, and user confidence. Overall, the deployment highlights the potential of embedded DataStage Assistants to enhance developer productivity and lower the entry barrier for enterprise data integration platforms.

Conclusion

ETL tools are crucial for organizing and assessing data. In this paper, we have shown how to simplify ETL flow creation by allowing users to describe transformations in natural language, making it easier for both beginners to learn and experts to streamline the process. We introduced a solution powered by LLMs for the creation of Extract, Transform, Load (ETL) flows and document exploration within a few-shot setting. This tool offers four key capabilities: (a) semantically analyzing user utterances to generate ETL operator recommendations, (b) inherently identifying interdependent relations by prompting for property value generation, (c) automatically generating and validating transformer stage expressions, and (d) providing information to users by semantically analyzing ETL documents to answer user queries.

The proposed framework was implemented and evaluated using the IBM watsonx.ai platform. To assess real-world utility, we conducted a user study involving both expert and novice users. The study highlighted substantial time savings, reduction in manual effort, and improved accuracy, with automation of transformer stage expressions proving especially beneficial for handling complex transformation logic. In addition, we experimented with multiple embedding models to support document exploration and ensure high retrieval relevance. Together, these results, supported by systematic evaluation of multiple LLMs, demonstrate the framework's ability to accelerate ETL development, reduce errors, and cater effectively to both novice and expert users.

Acknowledgments

We thank the entire IBM DataStage product and design team for their invaluable contributions to this work.

[†]https://community.ibm.com/community/user/blogs/shreya-sisodia/2025/06/26/ai-powered-datastage

References

Brachman, M.; Bygrave, C.; Chakraborti, T.; Chaudhary, A.; Ding, Z.; Dugan, C.; Gros, D.; Gschwind, T.; Johnson, J.; Laredo, J.; Miksovic, C.; Pan, Q.; Rai, P.; Ramalingam, R.; Scotton, P.; Surabathina, N.; and Talamadupula, K. 2022. A Goal-Driven Natural Language Interface for Creating Application Integration Workflows. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(11): 13155–13157.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H. P. D. O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Datacamp Team. 2023. A List of The 19 Best ETL Tools And Why To Choose Them. https://www.datacamp.com/blog/a-list-of-the-16-best-etl-\\tools-and-why-to-choose-them. Accessed: 2024-01-22.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805.

Dhamdhere, K.; McCurley, K. S.; Nahmias, R.; Sundararajan, M.; and Yan, Q. 2017. Analyza: Exploring Data with Conversation. In *Proceedings of the 22nd Annual Meeting of the Intelligent User Interfaces Community (ACM IUI 2017)*, 493–504. ACM.

ForsookComparison. 2025. Llama3 is better than Llama4.. is this anyone else's experience? https://www.reddit.com/r/LocalLLaMA/comments/1l6lp8x/llama3_is_better_than_llama4_is_this_anyone_elses/. Accessed: July 2025.

IBM, G. . B. I. 2025. https://huggingface.co/ibm-granite/granite-3.1-8b-instruct.

Jiang, L.; Cai, H.; and Xu, B. 2010. A domain ontology approach in the ETL process of data warehousing. In *ICEBE*.

Meta, A. 2024. Introducing meta llama 3: The most capable openly available llm to date. *Meta AI*, 2(5): 6.

Meta, A. 2025. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation. *https://ai.meta. com/blog/llama-4-multimodal-intelligence/, checked on*, 4(7): 2025.

Microsoft. n.d. Power Automate. https://www.microsoft.com/en-us/power-platform/products/power-automate. Accessed: 2024-01-22.

Nuenki. 2025. Llama 4 performs worse than Llama 3 at translation. https://nuenki.app/blog/llama_4_stats. Accessed: July 2025.

OpenAI; :; Agarwal, S.; Ahmad, L.; Ai, J.; Altman, S.; Applebaum, A.; Arbus, E.; Arora, R. K.; Bai, Y.; Baker, B.; Bao, H.; Barak, B.; Bennett, A.; Bertao, T.; Brett, N.; Brevdo, E.; Brockman, G.; Bubeck, S.; Chang, C.; Chen, K.; Chen, M.; Cheung, E.; Clark, A.; Cook, D.; Dukhan, M.; Dvorak, C.; Fives, K.; Fomenko, V.; Garipov, T.; Georgiev, K.; Glaese, M.; Gogineni, T.; Goucher, A.; Gross, L.; Guzman, K. G.; Hallman, J.; Hehir, J.; Heidecke, J.; Helyar, A.; Hu, H.; Huet, R.; Huh, J.; Jain, S.; Johnson, Z.; Koch, C.; Kofman, I.; Kundel, D.; Kwon, J.; Kyrylov, V.; Le, E. Y.; Leclerc, G.; Lennon, J. P.; Lessans, S.; Lezcano-Casado, M.; Li, Y.; Li, Z.; Lin, J.; Liss, J.; Lily; Liu; Liu, J.; Lu, K.; Lu, C.; Martinovic, Z.; McCallum, L.; McGrath, J.; McKinney, S.; McLaughlin, A.; Mei, S.; Mostovoy, S.; Mu, T.; Myles, G.; Neitz, A.; Nichol, A.; Pachocki, J.; Paino, A.; Palmie, D.; Pantuliano, A.; Parascandolo, G.; Park, J.; Pathak, L.; Paz, C.; Peran, L.; Pimenov, D.; Pokrass, M.; Proehl, E.; Qiu, H.; Raila, G.; Raso, F.; Ren, H.; Richardson, K.; Robinson, D.; Rotsted, B.; Salman, H.; Sanjeev, S.; Schwarzer, M.; Sculley, D.; Sikchi, H.; Simon, K.; Singhal, K.; Song, Y.; Stuckey, D.; Sun, Z.; Tillet, P.; Toizer, S.; Tsimpourlas, F.; Vyas, N.; Wallace, E.; Wang, X.; Wang, M.; Watkins, O.; Weil, K.; Wendling, A.; Whinnery, K.; Whitney, C.; Wong, H.; Yang, L.; Yang, Y.; Yasunaga, M.; Ying, K.; Zaremba, W.; Zhan, W.; Zhang, C.; Zhang, B.; Zhang, E.; and Zhao, S. 2025. gpt-oss-120b & gpt-oss-20b Model Card. arXiv:2508.10925.

Reddy, V. M.; and Jena, S. K. 2010. Active datawarehouse loading by tool based ETL procedure.

Scholak, T.; Schucher, N.; and Bahdanau, D. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.

Vassiliadis, P. 2009. A Survey of Extract-Transform-Load Technology. *IJDWAM*, 5: 1–27.

Vassiliadis, P.; Vagena, Z.; Skiadopoulos, S.; Karayannidis, N.; and Sellis, T. 2001. ARKTOS: towards the modeling, design, control and execution of ETL processes. *Information Systems*, 26(8): 537–561.

Wang, B.; Shin, R.; Liu, X.; Polozov, O.; and Richardson, M. 2021a. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers.

Wang, Y.; Wang, W.; Joty, S.; and Hoi, S. C. 2021b. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.

Zapier. 2023. Use the AI-powered Zap builder to generate Zaps (Beta). https://help.zapier.com/hc/en-us/articles/15703650952077/. Accessed: 2024-01-22.

Zhong, V.; Xiong, C.; and Socher, R. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *CoRR*.