# KGpipe: Generation of Pipelines for Data Integration into Knowledge Graphs

Marvin Hofer
ScaDS.AI Dresden/Leipzig
Leipzig, Germany
hofer@informatik.uni-leipzig.de

Erhard Rahm
ScaDS.AI Dresden/Leipzig
Leipzig, Germany
rahm@uni-leipzig.de

## ABSTRACT

We present KGpipe, a modular open-source framework for constructing and executing pipelines for integrating RDF, JSON, and text sources into knowledge graphs (KG). KGpipe supports heterogeneous execution backends, intermediate exchange formats, static pipeline validation, and can reuse existing implementations for integration tasks such as information extraction or ontology and entity matching, including LLM-based components. To demonstrate the framework's flexibility, we implement and comparatively evaluate multiple pipeline variants for the different source formats. The results show that structured RDF pipelines currently provide the most stable integration behavior, whereas JSON and text pipelines remain more sensitive to errors in mapping, extraction, and linking.

**VLDB Workshop Reference Format:**
Marvin Hofer and Erhard Rahm. KGpipe: Generation of Pipelines for Data Integration into Knowledge Graphs. VLDB 2026 Workshop: 15th International Workshop on Quality in Databases (QDB'26).

**VLDB Workshop Artifact Availability:**
The source code, data, and/or other artifacts have been made available at https://github.com/ScaDS/KGpipe.

## 1 INTRODUCTION

Knowledge graph (KG) construction and integration require combining heterogeneous data sources through multiple interacting tasks such as extraction, schema alignment, entity resolution, fusion, and validation [7]. In practice, these tasks are often implemented using different tools, programming languages, and execution environments, making integration pipelines difficult to construct, reproduce, and compare systematically. Existing solutions are frequently tailored to specific domains or source formats, resulting in ad hoc workflows with limited interoperability and reuse. Furthermore, existing tools for constructing KG integration pipelines are mostly not openly available and unable to reuse existing solutions for certain integration tasks [7].

The increasing diversity of source formats further complicates this problem. Structured RDF graphs, semi-structured JSON documents, and unstructured text sources require fundamentally different integration strategies. RDF integration primarily focuses on ontology alignment and entity matching, JSON integration requires schema interpretation and mapping, while text integration heavily depends on information extraction and entity linking. As a result, building reusable end-to-end KG integration pipelines remains challenging.

This paper presents KGpipe, a modular open-source framework for generating and executing heterogeneous KG integration pipelines. KGpipe supports pipeline composition across multiple execution backends, including Python tasks, Docker-based tools, and HTTP services. To improve interoperability between independently developed components, the framework introduces intermediate exchange formats together with static pipeline validation. This allows tasks operating on different data formats and implemented in different environments to be composed into reusable integration workflows.

To demonstrate the flexibility of the framework, we implement and analyze pipeline variants for RDF, JSON, and text integration settings using both traditional and LLM-based components. The pipelines are evaluated on the KGI-Bench [8] knowledge graph integration benchmark under a shared ontology and evaluation setting. The experiments illustrate characteristic differences between structured, semi-structured, and unstructured integration pipelines and highlight common challenges such as mapping ambiguity, entity fragmentation, and extraction errors.

In summary, this paper makes the following contributions:

- We present KGpipe, a modular open-source framework for constructing and executing heterogeneous KG integration pipelines.
- We introduce interoperable pipeline components with unified execution interfaces, intermediate exchange formats, and static validation mechanisms.
- We demonstrate the usefulness of the KGpipe framework by generating, executing and comparatively evaluating multiple pipeline variants for RDF, JSON, and text integration using both classical and LLM-based task implementations.

After a brief discussion of related work we give an overview of the KGpipe framework in Section 3. We then present the realized pipeline variants and their comparative evaluation (Sec. 4) before we conclude.

## 2 RELATED WORK

Research on KG integration spans specialized task-specific solutions as well as end-to-end KG construction systems. Numerous approaches address individual tasks such as information extraction [12], schema and ontology matching [1, 5], and entity resolution [3]. Tools such as JedAI [15] and Valentine [11] support


Proceedings of the VLDB Endowment. ISSN 2150-8097.

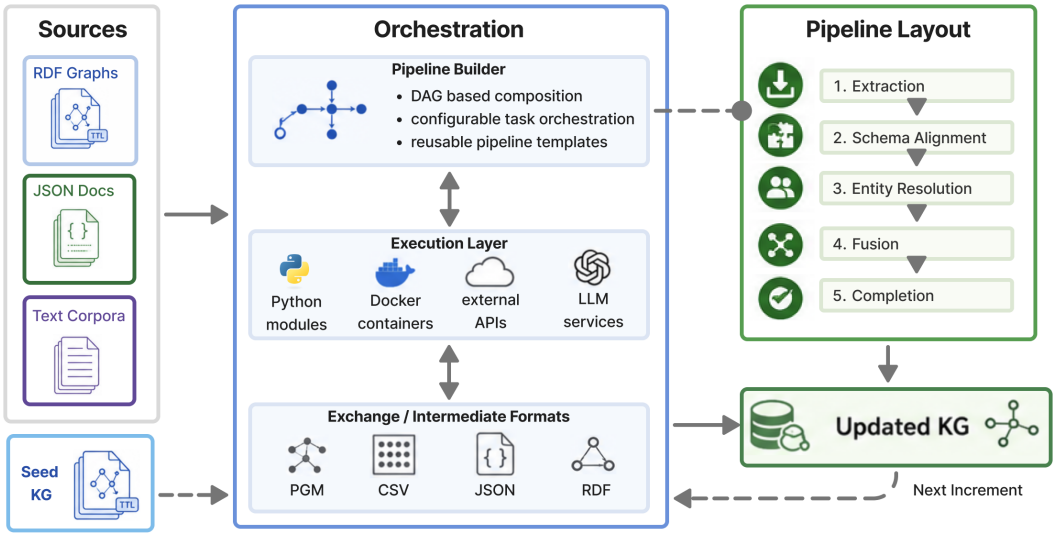


**Figure 1: Overview of the KGpipe framework for heterogeneous and incremental knowledge graph integration. Pipelines are constructed as modular DAGs of interoperable tasks spanning extraction, alignment, entity resolution, fusion, and validation stages. KGpipe supports RDF, JSON, and text sources through interchangeable execution backends and intermediate exchange formats, enabling systematic evaluation of alternative integration strategies across successive integration stages.**

configurable matching workflows, while recent work explores LLM-based support for supervised configuration of schema mapping and integration tasks [6, 14, 16].

Several systems support the determination of broader KG construction pipelines, including NELL [2], SAGA [9], XI [4], and Plumber [10]. However, these systems are mostly not openly available or focus on specific integration settings with handcrafted pipelines so that they are not generic enough to support the integration of different source formats. In our recent survey [7] we qualitatively evaluate available systems to determine end-to-end KG integration pipelines against a set of requirements and note a lack of powerful open, generic and modular tools that can reuse existing task-specific solutions within the pipelines.

KGpipe addresses this gap by providing a modular open-source framework for constructing and comparatively evaluating heterogeneous KG integration pipelines across RDF, JSON, and text sources. It allows the generation of several alternate pipelines for integrating a certain source into a given KG - the pipelines can then be comparatively evaluated, e.g. using the KGI-Bench approach [8].

## 3 KGPIPE FRAMEWORK

KGpipe implements a modular and extensible architecture that enables the specification, validation, and execution of KG integration pipelines. Each pipeline consists of tasks that can be executed in multiple ways, with clear input/output specifications that ensure compatibility across steps. To ensure reproducibility and easy debugging capabilities, our implementation uses file-based I/O between tasks. While this may introduce overhead compared to in-memory execution or direct streaming, it simplifies logging, inspection, and cross-language compatibility. All tasks are executed sequentially within a pipeline. Parallelism and optimization are reserved for future work.

Figure 1 illustrates the overall workflow of KGpipe for incremental KG construction and integration. Starting from a given seed KG the task is to determine a pipeline to integrate a new source of format RDF, JSON or text to generate an updated version of the KG. Each source is integrated by a separate pipeline.

Pipelines combine integration steps such as extraction, schema alignment, entity resolution, fusion, and validation. Each step is implemented as a modular task supporting different execution backends and interoperable exchange formats. Intermediate representations enable reusable workflows across independently developed components, while static validation ensures compatibility between consecutive tasks. The resulting integrated KG can then be evaluated using suitable quality metrics, as discussed in Section 4.

The remainder of this section is structured as follows. Section 3.1 introduces the formal pipeline specification and validation model. Section 3.2 describes the supported execution modes and backend abstraction. Section 3.3 discusses interoperability through exchange formats and transformation tasks. Section 3.4 presents the validation mechanisms and constraints, while the last section summarizes the task catalog used in this work.

### 3.1 Pipeline Specification

KGpipe represents each integration workflow as a directed acyclic graph (DAG) of interconnected tasks. A task corresponds to an integration operation such as extraction, schema matching, entity resolution, fusion, or validation. Each task specifies typed input and output interfaces together with its execution backend (Python, Docker, or HTTP service).

Pipelines are constructed by connecting compatible task outputs to subsequent task inputs. During pipeline construction, KGpipe performs static validation to ensure that adjacent tasks are compatible with respect to data formats, structural constraints, and intermediate exchange schemas. This validation prevents invalid task compositions before execution.

Formally, a pipeline is defined as a DAG $P = (T, E)$ where $T$ denotes the set of tasks and $E \subseteq T \times T$ represents directed dependencies between tasks. An edge $(t_i, t_j)$ exists if the output of task $t_i$ satisfies the input requirements of task $t_j$. The resulting graph defines both the execution order and data flow between integration stages.

To support interoperability across independently implemented components, KGpipe introduces lightweight intermediate exchange formats such as $JSON_{ER}$ for entity and relation matches and $JSON_{KE}$ for extracted triples and entity links. Transformation tasks can additionally be inserted automatically to bridge incompatible representations between pipeline stages.

## 3.2 Execution Modes

To accommodate the diversity of tools and runtimes used in KG integration, our framework abstracts task execution from the underlying implementation. We support three execution backends for tasks: Python tasks implemented using libraries such as `rdflib` or `transformers`, Docker-based tasks executed in isolated container environments for external tools or heterogeneous runtimes, and remote HTTP/LLM services accessed through RESTful APIs for scalable extraction, linking, or matching tasks.

These three backends are sufficient to cover the full spectrum of integration tasks we encounter: lightweight in-process operations (Python), encapsulated external tools or heterogeneous runtimes (Docker), and scalable remote services (HTTP). Supporting additional execution modes would largely duplicate functionality or add unnecessary complexity, whereas this set strikes a balance between flexibility, interoperability, and maintainability.

Pipelines are executed with a uniform interface regardless of backend, and temporary files are used for communication between tasks.

## 3.3 Task Interoperability

A central challenge is ensuring that task outputs are compatible with subsequent task inputs, particularly across heterogeneous execution backends and data formats. For example, some entity matching tools produce CSV outputs while fusion components expect structured JSON; information extraction tools may output raw triples without ontology alignment; and linker components may require specific JSON schemas or XML.

To address this, we introduce intermediate transformation tasks that convert outputs into the formats expected by subsequent components. These exchange tasks act as glue between independently developed modules. Our interoperability strategy is inspired by Plumber [10], but it extends this approach to support other task areas.

Currently, we define two intermediate exchange formats: $JSON_{ER}$, used for entity resolution tasks, representing entity or relation matches as:

**Table 1: Task catalog of `KGpipe`. OM=Ontology Matching, EM=Entity Matching, EF=Entity Fusion, TE=Triple Extraction, EL=Entity Linking, RL=Relation Linking, DM=Data Mapping, and TC=Type Completion. Backend types are Python (Py), Docker (D), and HTTP services (H).**

| Component | Tasks | Backend | I/O |
|---|---|---|---|
| PARIS | EM, OM | D | RDF → $JSON_{ER}$ |
| Valentine | OM | D | CSV → $JSON_{ER}$ |
| JedAI | EM | D | CSV → $JSON_{ER}$ |
| StanfordOpenIE | TE | D | TEXT → $JSON_{KE}$ |
| Spotlight | EL | D/H | TEXT → $JSON_{KE}$ |
| EmbeddingEL | EL | Py | $JSON_{KE}$ → $JSON_{KE}$ |
| EmbeddingRL | RL | Py | $JSON_{KE}$ → $JSON_{KE}$ |
| Tabularize | DM | Py | RDF → CSV |
| Json-to-RDF | DM | Py | JSON → RDF |
| GenerateRDF$_{KE}$ | DM | Py | $JSON_{KE}$ → RDF |
| FusionFirst | EF, TC | Py | RDF, $JSON_{ER}$ → RDF |
| SelectFirst | EF, TC | Py | RDF → RDF |
| LLMExtract | TE | H | TEXT → $JSON_{KE}$ |
| LLMMapping | DM | H | JSON, RDF → RDF |
| LLMMatcher | OM | H | RDF → $JSON_{ER}$ |

```
{ id1: $id # $number | $url
  id2: $id
  type: $str # "entity" | "relation"
  score: $float }
```

$JSON_{KE}$, used for knowledge extraction tasks, captures text, triples, and links, the latter representing mappings of text forms to URIs in the KG.

```
{ text: $str
  triples: [{head: $str, rel: $str, tail: $str}]
  links: [{form: $str, link: $id, score: $float}] }
```

## 3.4 Validation and Constraints

Each task specifies its expected input and output types, and during pipeline building, we apply static validation to ensure that consecutive tasks are compatible. This validation considers aspects such as file formats (e.g., TEXT, JSON, RDF), structural requirements like the presence of mandatory JSON keys or relation types, and, when available, schema compatibility between input and output. If a pipeline violates any of these constraints, it is rejected before execution.

To further support interoperability, these checks are aligned with the intermediate exchange formats introduced earlier ($JSON_{ER}$ and $JSON_{KE}$). By validating against these well-defined representations, we ensure that independently developed components can be safely composed into larger pipelines, even when they rely on different toolkits or programming languages.

## 3.5 Task Catalog

KGpipe organizes integration functionality into reusable tasks that can be composed into different pipeline configurations. Each task implements a specific integration capability such as ontology matching, entity resolution, extraction, mapping, linking, or fusion. To

**Table 2: Representative pipeline configurations for RDF, JSON, and text sources. The configurations illustrate selected task combinations rather than fixed workflows.**

**RDF pipelines**

| Variant | Matching | Fusion |
|---|---|---|
| $RDF_{base}$ | PARIS | FusionFirst |
| $RDF_{alt}$ | Tabularize + JedAI + Valentine | FusionFirst |
| $RDF_{llm}$ | LLMMatcher + PARIS | FusionFirst |

**JSON pipelines**

| Variant | Mapping | Matching/Linking | Fusion |
|---|---|---|---|
| $JSON_{base}$ | Json-to-RDF | PARIS | FusionFirst |
| $JSON_{alt}$ | Embedding | Embedding | SelectFirst |
| $JSON_{llm}$ | LLMMapping | PARIS | FusionFirst |

**Text pipelines**

| Variant | Extraction | Linking | Fusion |
|---|---|---|---|
| $TEXT_{base}$ | OpenIE | Spotlight + Embedding | SelectFirst |
| $TEXT_{alt}$ | OpenIE | Embedding | SelectFirst |
| $TEXT_{llm}$ | LLMExtract | Embedding | SelectFirst |

support interoperability across heterogeneous implementations, all tasks expose a unified interface consisting of typed input and output formats together with an execution backend.

Table 1 summarizes the task catalog of `KGpipe` used in this work. The catalog combines external integration tools, custom Python components, and LLM-based services under a shared execution abstraction. Tasks can operate on RDF graphs, JSON documents, tabular CSV data, plain text, or the intermediate exchange formats introduced earlier ($JSON_{ER}$ and $JSON_{KE}$).

The catalog includes several established tools. PARIS [17] performs ontology and entity matching directly on RDF graphs, while JedAI [15] and Valentine [11] support entity matching and schema matching on tabular data. For text-based extraction, Stanford OpenIE [12] generates surface-form triples and DBpedia Spotlight [13] performs entity linking against the target KG.

In addition, KGpipe provides several custom integration tasks implemented in Python. These include transformation tasks such as *Tabularize*, *Json-to-RDF*, and *GenerateRDF*$_{KE}$. The embedding-based entity and relation linking tasks use text embeddings over entity labels, relation names, and extracted surface forms to identify semantically similar KG elements. Fusion tasks such as *FusionFirst* and *SelectFirst* combine integrated graphs while additionally performing type completion based on the ontology domain and range axioms.

Finally, KGpipe supports LLM-based integration tasks through HTTP services. These tasks include *LLMExtract* for text-based triple extraction, *LLMMapping* for ontology-aware RDF generation from JSON documents, and *LLMMatcher* for LLM-assisted ontology matching. Since all tasks share compatible I/O contracts, they can be substituted or recombined within different pipeline layouts without modifying the overall execution framework.

# 4 EVALUATION

To demonstrate the usefulness of the KGpipe framework, we generated and executed multiple pipeline variants for their comparative evaluation. In particular, we compare pipelines operating on structured RDF, semi-structured JSON, and unstructured text sources under a shared KG ontology and a common evaluation setting. This should enable us to study how source characteristics and the design of the KGpipe-generated pipelines influence the quality of the resulting knowledge graph.

KGpipe supports RDF, JSON, and text sources through configurable pipeline layouts composed of interchangeable integration tasks. For each source type, we implement three representative variants: a baseline pipeline, an alternative pipeline using different matching or linking strategies, and an LLM-assisted pipeline replacing selected components with large language models. These pipelines are intended as illustrative configurations rather than fixed workflows, since tasks can be substituted or recombined depending on the integration setting. Table 2 summarizes the implemented pipeline configurations and their selected task implementations. For each source format, the baseline, alternative, and LLM-assisted variants differ in the task implementation used at the main format-specific integration step. For RDF, $RDF_{base}$ uses PARIS for ontology and entity matching, $RDF_{alt}$ converts the graphs to tabular form and applies JedAI and Valentine, and $RDF_{llm}$ replaces ontology matching with an LLM-based matcher while retaining PARIS for entity matching. For JSON, $JSON_{base}$ first converts JSON to RDF, $JSON_{alt}$ performs embedding-based entity and relation linking, and $JSON_{llm}$ directly generates ontology-aligned RDF using an LLM. For text, $TEXT_{base}$ combines OpenIE with DBpedia Spotlight and embedding-based relation linking, $TEXT_{alt}$ uses embedding-based linking for both entities and relations, and $TEXT_{llm}$ replaces OpenIE with LLM-based triple extraction.

The experiments use the *KGI-Bench* benchmark datasets for the movie domain (KGI-Bench-Movie), consisting of a seed KG, a reference KG (as the ground truth for the updated KG), and multiple overlapping source splits derived from DBpedia [8]. The benchmark contains film, person, and company entities connected through a shared ontology with 25 relations and attributes. There are three source files (in any of the three formats RDF, JSON, text) that have to be integrated one after the other into the seed KG to account for the incremental nature of KG updates. We primarily use the 10k films variant, where the seed KG contains 123,686 facts and 19,527 entities (film, person, companies), while the final reference KG contains 336,002 facts and 47,706 entities. Due to the substantially higher runtime costs of LLM-based components, the $JSON_{llm}$and $TEXT_{llm}$configurations were executed only on the reduced 1k benchmark variant, where the seed KG contains 16,417 facts, and 2,793 entities, and the complete reference KG contains 63,359 facts and 8,935 entities. This reduced setup is also reflected in Figure 2, where the corresponding pipelines are omitted from the entity and triple growth analysis.

To ensure comparability across pipeline configurations, we use fixed matching and linking thresholds throughout all experiments, determined through preliminary experiments. PARIS uses a similarity threshold of 0.95 for entity matches and 0.5 for relation matches. JedAI applies a matching threshold of 0.5, while Valentine uses 0.1

**Table 3: Incremental integration quality across heterogeneous KG pipelines after successive integration stages. Values are reported as Stage 1 / Stage 2 / Stage 3 for coverage, correctness, and semantic consistency. Rows evaluated on the 1k benchmark variant are explicitly marked and should only be compared with scale-matched rows. The table also summarizes characteristic integration behavior and dominant failure modes observed for RDF-, JSON-, and text-based pipelines. Dominant behavior and main failure source are reported once per pipeline design and are not repeated for scale-matched reruns.**

| Pipeline | Coverage | Correctness | Consistency | Dominant Behavior | Main Failure Source |
|---|---|---|---|---|---|
| $RDF_{base}$ | 0.69 / 0.69 / 0.70 | 0.89 / 0.88 / 0.89 | 1.00 / 1.00 / 1.00 | Stable structured integration | Minor fusion mismatches |
| $RDF_{alt}$ | 0.59 / 0.58 / 0.58 | 0.77 / 0.76 / 0.74 | 0.99 / 0.99 / 0.99 | Recall decreases during tabular mapping | Entity fragmentation |
| $RDF_{llm}$ | 0.94 / 0.93 / 0.92 | 0.96 / 0.95 / 0.95 | 1.00 / 1.00 / 1.00 | Strongest overall stability | Runtime overhead |
| $JSON_{base}$ | 0.63 / 0.63 / 0.62 | 0.86 / 0.85 / 0.85 | 0.96 / 0.95 / 0.94 | Moderate degradation over stages | Schema ambiguity |
| $JSON_{alt}$ | 0.29 / 0.28 / 0.28 | 0.56 / 0.56 / 0.55 | 0.98 / 0.97 / 0.97 | Embedding mappings accumulate errors | Incorrect relation linking |
| $JSON_{base}$ (1k) | 0.78 / 0.79 / 0.77 | 0.89 / 0.89 / 0.89 | 0.95 / 0.93 / 0.93 | — | — |
| $JSON_{llm}$ (1k) | 0.61 / 0.62 / 0.60 | 0.75 / 0.76 / 0.76 | 1.00 / 0.99 / 0.99 | Better consistency but unstable outputs | LLM variability |
| $TEXT_{base}$ | 0.01 / 0.01 / 0.01 | 0.22 / 0.23 / 0.24 | 0.99 / 0.99 / 0.99 | Severe extraction loss | Entity linking failures |
| $TEXT_{alt}$ | 0.03 / 0.03 / 0.03 | 0.16 / 0.17 / 0.18 | 0.95 / 0.92 / 0.89 | Increasing semantic corruption | Relation direction errors |
| $TEXT_{base}$ (1k) | 0.01 / 0.01 / 0.01 | 0.18 / 0.17 / 0.18 | 1.00 / 1.00 / 1.00 | — | — |
| $TEXT_{llm}$ (1k) | 0.04 / 0.04 / 0.04 | 0.22 / 0.24 / 0.24 | 0.92 / 0.87 / 0.82 | Slightly improved extraction recall | Hallucinated triples |

for schema matching. DBpedia Spotlight performs entity linking at a confidence threshold of 0.8. The embedding-based entity and relation linkers (*EmbeddingEL* and *EmbeddingRL*) both use a similarity threshold of 0.8. For LLM-based tasks, we used OpenAI's `gpt-5-mini-2025-08-07` model snapshot across all experiments to balance runtime, cost, and output quality.

**Execution cost.** We additionally measured end-to-end wall-clock runtime and peak memory consumption for each pipeline. On the 10k benchmark, the RDF pipelines required 66 s for $RDF_{base}$, 6,512 s for $RDF_{alt}$, and 196 s for $RDF_{llm}$, with peak memory consumption of 6.3 GB, 44.5 GB, and 6.1 GB, respectively. The JSON and text LLM pipelines were evaluated only on the smaller 1k benchmark because of their substantially higher execution costs: $JSON_{llm}$ required 8,749 s and 4.8 GB, while $TEXT_{llm}$ required 2,006 s and 10.4 GB. Their corresponding API costs were approximately €9.90 and €2.30 per complete three-stage run; $RDF_{llm}$ cost approximately €0.10. These measurements explain the use of the reduced benchmark for the JSON and text LLM variants. Since these variants were executed on a different benchmark size, their runtimes should not be interpreted as direct speed comparisons with the corresponding 10k baseline pipelines. Runtime and API cost may also vary with hardware, network latency, service load, and provider pricing.

## 4.1 Metrics

We evaluate the generated KGs using the three complementary quality dimensions defined in KGI-Bench [8]: coverage, correctness, and consistency. Coverage measures how completely source information is represented in the integrated KG. Correctness evaluates whether integrated entities and triples correspond to the reference KG without introducing duplicates or incorrect facts. Consistency measures adherence to ontology constraints such as valid domain/range usage, relation direction, and datatype correctness.

**Metric computation.** Let $KG_i$ denote the graph produced after integration stage $i$, and let $KG_R^{(i)}$ denote the corresponding reference graph containing the information expected up to that stage. Entities and triples already present in the seed KG are excluded from both graphs before evaluation. Because equivalent entities may use different identifiers, an alignment $A_i$ between entities in $KG_i$ and $KG_R^{(i)}$ is derived using sentence-transformer-based similarity over entity labels. Literal values are also compared using fuzzy equality and datatype- or format-specific normalization. A produced triple matches a reference triple if their subjects are aligned, their predicates are identical, and their objects are either aligned entities or equivalent literals after datatype and lexical normalization. Matching is binary: a triple satisfying only some of these conditions receives no partial credit.

Let $\mathrm{Cov}_E$ and $\mathrm{Cov}_T$ denote the fractions of expected reference entities and triples represented in $KG_i$, respectively. Let $\mathrm{Corr}_E$ and $\mathrm{Corr}_T$ denote the fractions of produced entities and triples that correspond to the reference graph. Duplicate produced entities aligned to the same reference entity receive only one correct match and therefore reduce entity correctness. The aggregate values reported in Table 3 are

$$\mathrm{Coverage}_i = \frac{\mathrm{Cov}_E(KG_i) + \mathrm{Cov}_T(KG_i)}{2}$$

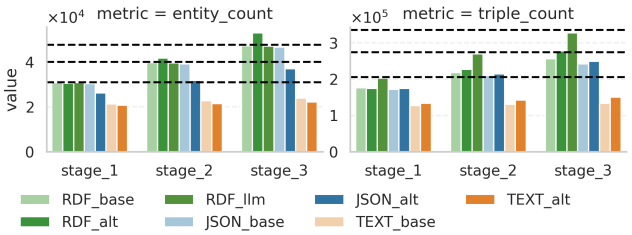


**Figure 2: Evolution of entity and triple counts across incremental integration stages. Dashed lines indicate reference KG sizes. $JSON_{llm}$ and $TEXT_{llm}$, the LLM-based pipelines, were executed on the smaller 1k dataset variant due to higher runtime costs and are therefore omitted.**

$$\text{Correctness}_i = \frac{\text{Corr}_E(KG_i) + \text{Corr}_T(KG_i)}{2}.$$

Consistency is computed independently of the reference graph. For each applicable ontology constraint $c$, we determine the violation ratio $v_c/n_c$, where $v_c$ is the number of violating entities, relations, or literals and $n_c$ is the number of corresponding evaluation units. We consider disjoint-class, domain, range, relation-direction, datatype, and literal-format violations. The reported consistency value is the arithmetic mean of their compliance scores:

$$\text{Consistency}_i = \frac{1}{|C|} \sum_{c \in C} \left(1 - \frac{v_c}{n_c}\right).$$

For example, a relation with a correct subject but an incorrectly linked object contributes neither to triple coverage nor triple correctness, while a relation whose subject and object types violate its declared domain or range contributes one corresponding consistency violation. Full definitions of the alignment procedure, entity- and triple-level metrics, literal normalization, and individual ontology-violation measures are provided in the companion KGI-Bench specification [8].

All metrics are computed incrementally after each integration stage, enabling analysis of how errors propagate and accumulate during sequential KG updates. Higher coverage and correctness values indicate more complete and semantically accurate integration results, while higher consistency values indicate stronger adherence to the target ontology and fewer structural violations.

In the following evaluation, we focus on comparative quality differences between RDF-, JSON-, and text-based pipelines to analyze how source structure and pipeline design influence KG quality.

## 4.2 Data Quality Results and Error Propagation

We analyze the pipeline results from a data-quality perspective along three dimensions: completeness, correctness, and semantic consistency. In addition, we distinguish recurring failure modes according to the pipeline stage in which they originate: representation and mapping errors, entity resolution and linking errors, extraction errors, and fusion-related propagation errors. Because integration is incremental, local errors introduced at one stage may persist or be amplified in later KG updates.

Figure 2 illustrates the evolution of entity and triple counts across incremental integration stages, while Table 3 summarizes coverage, correctness, and consistency for all evaluated pipelines. Since integration is performed incrementally, errors introduced during early stages may propagate into subsequent updates. The evaluation therefore analyzes not only final KG quality, but also the robustness of different pipeline designs under repeated integration steps.

The observed failures can be grouped into four recurring data-quality categories. First, representation and mapping errors arise when structural information is lost or ambiguous, as in RDF tabularization and JSON schema interpretation. Second, identity and linking errors include entity fragmentation, missed entity matches, and incorrect entity or relation links. Third, extraction errors occur mainly for text sources and include missing, hallucinated, or directionally incorrect triples. Fourth, propagation errors arise when incorrect mappings or entities introduced in early increments are reused during later fusion steps. These categories affect completeness, correctness, and consistency in different ways and explain the source-dependent behavior summarized in Table 3.

The RDF-based pipelines exhibit the most stable integration behavior across all stages. Since RDF sources already provide explicit graph structure and semantic relations, the integration problem is largely reduced to ontology and entity alignment. Consequently, $\text{RDF}_{\text{base}}$ and $\text{RDF}_{\text{llm}}$ maintain full semantic consistency throughout all stages, indicating that structured RDF representations substantially reduce schema ambiguity and invalid relation generation.

Coverage and correctness remain comparatively stable for the RDF pipelines because most entities and relations can already be aligned during early integration stages. $\text{RDF}_{\text{llm}}$ achieves the strongest overall performance in both coverage and correctness. The LLM-assisted ontology matching improves relation alignment between source and seed graphs, reducing missed mappings while preserving semantic consistency. In contrast, $\text{RDF}_{\text{alt}}$ shows decreasing coverage and correctness over successive stages. The conversion of RDF graphs into tabular representations introduces information loss and entity fragmentation, which negatively affects both entity matching and downstream fusion quality. Once fragmented entities are introduced into the KG, later integration stages increasingly reuse these incomplete representations, further reducing matching quality.

The JSON pipelines exhibit moderate degradation compared to RDF. Unlike RDF sources, JSON documents do not explicitly encode ontology semantics, requiring additional schema interpretation and mapping steps. Consequently, integration quality depends strongly on the selected mapping strategy.

$\text{JSON}_{\text{base}}$ achieves comparatively robust behavior by first lifting JSON data into RDF and subsequently applying graph-based alignment. Coverage and correctness remain relatively stable across stages, although generic relation construction still introduces schema ambiguity that accumulates over time. $\text{JSON}_{\text{alt}}$ performs substantially worse in both coverage and correctness. The embedding-based matching strategy frequently links semantically similar but structurally incompatible entities or relations, leading to accumulated mapping errors and fragmented entity representations. These incorrect mappings propagate into later integration stages and increasingly affect downstream fusion decisions, causing persistent quality degradation. $\text{JSON}_{\text{llm}}$ improves semantic consistency by generating ontology-aware RDF triples directly with an LLM, but output variability still causes fluctuations in correctness and coverage across stages. While ontology-aware generation reduces structural violations, occasional hallucinated or incomplete mappings still introduce instability during incremental updates.

The text-based pipelines remain the most challenging integration setting. In contrast to RDF and JSON sources, text pipelines must first recover structured information through extraction and linking, making errors propagate across multiple dependent stages.

All text pipelines achieve substantially lower coverage due to incomplete extraction and linking failures. In $\text{TEXT}_{\text{base}}$, OpenIE and Spotlight frequently fail to recover entities and relations required for integration into the target KG, resulting in almost no growth in coverage across stages. $\text{TEXT}_{\text{alt}}$ replaces symbolic linking with embedding-based matching, but relation direction errors and semantically incorrect matches increasingly corrupt the KG

during incremental updates. As a result, correctness and consistency continuously decrease across stages because invalid relations and incorrectly aligned entities accumulate in the integrated graph. $\text{TEXT}_{\text{llm}}$ slightly improves coverage through LLM-based extraction, but hallucinated triples and incorrect relation generation reduce semantic consistency over time. The generated triples often appear structurally plausible while violating ontology semantics, making such errors difficult to detect during later fusion stages.

The incremental setting reveals an important data-quality effect: errors are not independent across stages. A missed match may create a duplicate entity, which then reduces the likelihood of correct matching in later increments. Similarly, an incorrect relation mapping may be reused during subsequent fusion, causing a local schema error to affect multiple later triples. This explains why RDFalt, JSONalt, and TEXTalt show progressive degradation even though their task configurations and thresholds remain fixed.

Overall, the experiments demonstrate that integration robustness is strongly influenced by source structure and the number of required semantic interpretation steps. Structured RDF integration remains comparatively stable, whereas JSON and text pipelines are substantially more sensitive to mapping ambiguity, extraction noise, and semantic linking errors. The results further show that incremental KG integration amplifies early-stage errors, making robust matching and validation strategies essential for maintaining long-term KG quality.

## 5 CONCLUSION

We presented KGpipe, a modular open-source framework for constructing and executing different knowledge graph integration pipelines across RDF, JSON, and text sources. By supporting interchangeable task implementations and multiple execution backends, KGpipe can utilize and reuse already proven solutions and also integrate LLM functionality. KGpipe is a generic tool that can be used for knowledge graphs and data sources of virtually any domain. It supports a comparative evaluation of generated pipelines to help find the best solutions. The presented experiments on KGI-Bench showed the usability of KGpipe and illustrated how different pipeline designs and source formats affect integration quality, robustness, and semantic consistency, while also highlighting the strengths and limitations of LLM-based components in KG integration workflows.

An important direction for future work is the automated optimization of task-level hyperparameters across complete integration pipelines. While KGpipe currently supports explicit configuration of component parameters, these values are selected externally and remain fixed during pipeline execution. Future work will investigate pipeline-level optimization that jointly considers integration quality, runtime, and execution cost.

## ACKNOWLEDGMENTS

The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany and by the Sächsische Staatsministerium für Wissenschaft Kultur und Tourismus in the program Center of Excellence for AI-research "Center for Scalable Data Analytics and Artificial Intelligence Dresden/Leipzig", project identification number: ScaDS.AI.

## REFERENCES

[1] Zohra Bellahsene, Angela Bonifati, and Erhard Rahm. 2011. *Schema Matching and Mapping*. Springer.

[2] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. 2010. Toward an Architecture for Never-Ending Language Learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, Maria Fox and David Poole (Eds.). AAAI Press, 1306–1313. https://doi.org/10.1609/AAAI.V24I1.7519

[3] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2021. An Overview of End-to-End Entity Resolution for Big Data. *ACM Comput. Surv.* 53, 6 (2021), 127:1–127:42. https://doi.org/10.1145/3418896

[4] Philippe Cudré-Mauroux. 2020. Leveraging knowledge graphs for big data integration: the XI pipeline. *Semantic Web* 11, 1 (2020), 13–17.

[5] Juliana Freire, Grace Fan, Benjamin Feuer, Christos Koutras, Yurong Liu, Eduardo Peña, and Eden Wu. 2025. Large Language Models for Data Discovery and Integration: Challenges and Opportunities. *IEEE Data Eng. Bull* 49, 1 (2025), 3–31.

[6] Marvin Hofer, Johannes Frey, and Erhard Rahm. 2024. Towards self-configuring Knowledge Graph Construction Pipelines using LLMs - A Case Study with RML. In *Proceedings of the 5th International Workshop on Knowledge Graph Construction co-located with 21th Extended Semantic Web Conference (ESWC 2024), Hersonissos, Greece, May 27, 2024 (CEUR Workshop Proceedings)*, David Chaves-Fraga, Anastasia Dimou, Ana Iglesias-Molina, Umutcan Serles, and Dylan Van Assche (Eds.), Vol. 3718. CEUR-WS.org. https://ceur-ws.org/Vol-3718/paper6.pdf

[7] Marvin Hofer, Daniel Obraczka, Alieh Saeedi, Hanna Köpcke, and Erhard Rahm. 2024. Construction of Knowledge Graphs: Current State and Challenges. *Inf.* 15, 8 (2024), 509. https://doi.org/10.3390/INFO15080509

[8] Marvin Hofer and Erhard Rahm. 2026. Evaluation of Pipelines for Data Integration into Knowledge Graphs. arXiv:2605.22304 [cs.AI] https://arxiv.org/abs/2605.22304

[9] Ihab F. Ilyas, Theodoros Rekatsinas, Vishnu Konda, Jeffrey Pound, Xiaoguang Qi, and Mohamed A. Soliman. 2022. Saga: A Platform for Continuous Construction and Serving of Knowledge at Scale. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 2259–2272. https://doi.org/10.1145/3514221.3526049

[10] Mohamad Yaser Jaradeh, Kuldeep Singh, Markus Stocker, Andreas Both, and Sören Auer. 2021. Better Call the Plumber: Orchestrating Dynamic Information Extraction Pipelines. In *Web Engineering - 21st International Conference, ICWE 2021, Biarritz, France, May 18-21, 2021, Proceedings (Lecture Notes in Computer Science)*, Marco Brambilla, Richard Chbeir, Flavius Frasincar, and Ioana Manolescu (Eds.), Vol. 12706. Springer, 240–254. https://doi.org/10.1007/978-3-030-74296-6_19

[11] Christos Koutras, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Jerry Brons, Marios Fragkoulis, Christoph Lofi, Angela Bonifati, and Asterios Katsifodimos. 2021. Valentine: Evaluating Matching Techniques for Dataset Discovery. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 468–479. https://doi.org/10.1109/ICDE51399.2021.00047

[12] José-Lázaro Martínez-Rodríguez, Ivan López-Arévalo, and Ana B. Ríos-Alvarado. 2018. OpenIE-based approach for Knowledge Graph construction from text. *Expert Syst. Appl.* 113 (2018), 339–355. https://doi.org/10.1016/J.ESWA.2018.07.017

[13] Pablo N. Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. 2011. DBpedia spotlight: shedding light on the web of documents. In *Proceedings the 7th International Conference on Semantic Systems, I-SEMANTICS 2011, Graz, Austria, September 7-9, 2011 (ACM International Conference Proceeding Series)*, Chiara Ghidini, Axel-Cyrille Ngonga Ngomo, Stefanie N. Lindstaedt, and Tassilo Pellegrini (Eds.). ACM, 1–8. https://doi.org/10.1145/2063518.2063519

[14] Konstantinos Nikoletos, Vasilis Efthymiou, George Papadakis, and Kostas Stefanidis. 2025. Auto-Configuring Entity Resolution Pipelines. *IEEE Access* 13 (2025), 155367–155384. https://doi.org/10.1109/ACCESS.2025.3600541

[15] George Papadakis, Leonidas Tsekouras, Emmanouil Thanos, Nikiforos Pittaras, Giovanni Simonini, Dimitrios Skoutas, Paul Isaris, George Giannakopoulos, Themis Palpanas, and Manolis Koubarakis. 2020. JedAI$^3$ : beyond batch, blocking-based Entity Resolution. In *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*, Angela Bonifati, Yongluan Zhou, Marcos Antonio Vaz Salles, Alexander Böhm, Dan Olteanu, George H. L. Fletcher, Arijit Khan, and Bin Yang (Eds.). OpenProceedings.org, 603–606. https://doi.org/10.5441/002/EDBT.2020.74

[16] Aaron Steiner and Christian Bizer. 2026. Automatic End-to-End Data Integration using Large Language Models. *CoRR* abs/2603.10547 (2026). https://doi.org/10.48550/ARXIV.2603.10547 arXiv:2603.10547

[17] Fabian M. Suchanek, Serge Abiteboul, and Pierre Senellart. 2011. PARIS: Probabilistic Alignment of Relations, Instances, and Schema. *Proc. VLDB Endow.* 5, 3 (2011), 157–168. https://doi.org/10.14778/2078331.2078332