Configuration and Optimization of Knowledge Graph Integration Pipelines

Marvin Hofer[1,2,*], Victor Christen[1,2,†] and Erhard Rahm[1,2]

[1]Institute of Computer Science, Leipzig University, Germany, https://cs.uni-leipzig.de
[2]Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI), Dresden/Leipzig, Germany, https://scads.ai

Abstract

Knowledge graph construction commonly relies on multi-stage pipelines that combine extraction, matching, linking, and fusion tasks. Each task may provide multiple implementations and parameter settings, yielding a large configuration space in which task-specific decisions can strongly affect end-to-end graph quality. In practice, task and pipeline configurations are often selected manually, leaving many promising pipeline variants unexplored.

We formulate the configuration of KG integration pipelines as a budget-constrained optimization problem over task implementations and parameters. We introduce a quality-aware search framework that iteratively generates valid configurations, executes the corresponding pipelines, evaluates the resulting knowledge graphs, and uses quality feedback to guide subsequent choices. The framework enables a direct comparison of search strategies under a common evaluation protocol, including random search, Bayesian optimization, random neighborhood search, and a structure-aware heuristic refinement strategy. The evaluation on structured RDF and text-based KG integration scenarios shows that high-quality configurations can be identified under limited evaluation budgets, while the relative effectiveness of the search strategies depends on the structure and quality landscape of the configuration space.

Keywords

Knowledge Graph, Data Integration, Data Quality

1. Introduction

Knowledge graphs (KGs) have become a central paradigm for integrating heterogeneous data into a unified and semantically rich representation. Integrating sources into high-quality KGs typically requires multi-stage pipelines that combine tasks such as information extraction, schema alignment, entity resolution, and data fusion. Over the past years, a large number of methods and tools have been developed for each of these tasks, including both traditional approaches and recent large language model (LLM)-based techniques.

Despite this progress, building effective KG integration pipelines remains challenging. In practice, pipelines are often manually designed and configured for specific domains or datasets. This leads to a proliferation of ad hoc solutions that are difficult to reuse, compare, or generalize. More importantly, pipeline design involves a large number of decisions, including the selection of task implementations and the tuning of their parameters. These decisions interact across pipeline stages and directly affect the quality of the resulting KG.

Recent work, including our own KGpipe framework, has addressed the problem of systematically defining, executing, and evaluating KG integration pipelines [1]. KGpipe enables the composition of pipelines from reusable components and provides a benchmark for comparing different pipeline designs. However, existing approaches still rely on manually selected configurations and evaluate pipelines only after they have been constructed. As a result, they do not address the central question of how to automatically identify high-quality pipeline configurations.

DMKG'26: 2nd International Workshop on Data Management for Knowledge Graphs, October 2026, Bari, Italy
*Corresponding author.
†These authors contributed equally.
hofer@informatik.uni-leipzig.de (M. Hofer); christen@informatik.uni-leipzig.de (V. Christen); rahm@informatik.uni-leipzig.de (E. Rahm)
0000-0003-4667-5743 (M. Hofer); 0000-0001-7175-7359 (V. Christen); 0000-0002-2665-1114 (E. Rahm)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-09

CEUR-WS.org/Vol-4322/paper_24.pdf

This paper addresses this gap by introducing a *quality-aware configuration* approach for KG integration pipelines. We formulate pipeline configuration as an optimization problem over task implementations and parameter settings within a given pipeline layout. Unlike prior work that focuses on individual tasks or post-hoc evaluation, we explicitly close the loop between pipeline configuration, execution, and quality assessment.

Building on KGpipe, we propose a search-based framework that iteratively generates pipeline configurations, executes them, and updates the search based on estimated quality scores. The approach optimizes an aggregate objective derived from complementary KG quality dimensions, including correctness, coverage, and consistency.

We evaluate our framework on structured and text-based KG integration scenarios. The results show that high-quality configurations can be found under limited evaluation budgets, while the effectiveness of individual search strategies depends on the structure and quality landscape of the configuration space.

In summary, this paper makes the following contributions:

- We formalize quality-aware configuration of KG integration pipelines as a budget-constrained optimization problem over task implementations and parameter settings.
- As a first step toward automated, quality-driven configuration of KG integration pipelines we test different search approaches that iteratively generate, execute, evaluate, and update pipeline configurations based on end-to-end KG quality signals.
- We implement the method within the KGpipe framework and empirically evaluate it on structured and text-based KG integration scenarios, analyzing configuration quality and search efficiency.

By reducing the number of expensive end-to-end pipeline executions required to identify strong configurations, the approach contributes to the systematic benchmarking and resource-efficient management of KG integration workflows. The source code, experimental configurations, and data used in this study are publicly available at [1].

2. Related Work

Knowledge graph (KG) construction has been widely studied from perspectives including integration pipelines, automated configuration, and quality assessment. Our work focuses on using end-to-end KG quality to guide the configuration of multi-stage integration pipelines. KG construction is typically realized as a multi-stage pipeline that transforms heterogeneous sources into a unified graph representation. Key stages include information extraction, schema alignment, entity resolution (ER), and data fusion [2]. These stages are interdependent, with errors propagating across stages and downstream quality relying on upstream decisions. Research has focused on improving individual tasks, such as triple extraction [3], entity and relation linking [4], and coreference resolution [5]. For multi-source KGs, schema alignment [6], ER [7, 8, 9], and data fusion [10] are essential. Hybrid approaches like PARIS [11] and SIGMa [12] address multiple tasks simultaneously. Recent studies also explore LLMs for ontology construction, knowledge extraction, and fusion [13], though these methods often lack flexibility and user configurability.

Automated Configuration. To address configuration complexity, frameworks such as Plumber [14] support pipeline orchestration, but are limited to a single data source. Nikoletos et al. [15] address the configuration of ER pipelines with fixed steps and parameter spaces, using strategies with or without ground-truth data. Steiner et al. [16] propose an automated end-to-end data integration pipeline in which LLMs support schema matching, normalization, entity matching, and data fusion. Their entity-matching component uses active learning with an LLM as an oracle, similar to [17], while fusion configurations are assessed using generated validation sets.

In machine learning, hyperparameter optimization aims to identify high-quality configurations efficiently [18, 19, 20]. In contrast to grid or random search, Bayesian optimization uses observations

[1] https://github.com/ScaDS/KGpipe/tree/main/experiments/param-opti

from previously evaluated configurations to guide subsequent selections [19]. Frameworks such as Optuna [21] additionally support the dynamic definition of conditional search spaces.

KG Quality. Assessing the quality of KGs is an active research area and an integral component of configuring KG integration pipelines. Prior work has proposed dimensions and metrics commonly covering accuracy, coverage, and consistency [2]. Evaluation approaches range from partial gold-standard comparison [22, 23] and constraint validation [24] to reference-independent methods [25].

Marchesin et al. [23] iteratively select samples to reduce labeling cost while bounding the estimation error. Zhang et al. [26] tailor sample selection to users' specific usage requirements. Shami et al. [25] investigate LLM-based judges supported by externally retrieved evidence. These approaches provide quality estimates for constructed graphs, but do not use end-to-end quality feedback to guide the configuration of multi-stage integration pipelines.

Limitations. Automated configuration has been studied extensively in hyperparameter optimization and AutoML, while related work in data integration has mainly focused on individual tasks or fixed workflows, particularly ER [15]. In contrast, KG integration pipelines combine heterogeneous components whose implementation and parameter choices interact across stages and require expensive end-to-end evaluation. Our work therefore extends automated configuration to the joint search over task implementations and parameters in multi-stage KG integration pipelines, using end-to-end graph quality as the optimization objective.

3. Problem Formulation

We consider the configuration of a KG integration pipeline for a given integration scenario. Let $S = (S_1, \ldots, S_n)$ denote the input sources, KG_0 an optional seed KG, and $\mathcal{O}$ the target ontology. The pipeline integrates the information from S into KG_0 and produces a resulting KG.

We assume a fixed high-level pipeline layout $L = (T_1, \ldots, T_m)$ where each T_j denotes an integration task, such as information extraction, ontology matching, entity matching, entity linking, or data fusion. Optimizing the pipeline layout itself is outside the scope of this work. Each task T_j can be instantiated by one of several available implementations, $\mathcal{I}_j = \{I_{j,1}, \ldots, I_{j,k}\}$.

An implementation $I_{j,k} \in \mathcal{I}_j$ is configurable by a set of parameters $\theta_{j,k}$ with domain $\Theta_{j,k}$. A parameter of $\theta_{j,k}$ can be numeric or categorical. Moreover, the parameter space can differ according to various implementations, e.g., one entity-matching implementation requires a similarity threshold, whereas another may not expose any parameters.

A pipeline configuration is therefore defined as follows:

$\lambda = \big((I_{1,k}, \theta_{1,k}), \ldots, (I_{m,l}, \theta_{m,l})\big)$, where $I_{m,l} \in \mathcal{I}_m$ and $\theta_{m,l} \in \Theta_{m,l}$ for task T_m instantiated by an implementation $I_{m,l}$. The set of all valid configurations for layout L is denoted by Λ_L. A configuration is valid if the selected implementations are compatible with respect to their required inputs and produced outputs.

The execution P of a pipeline configuration λ according to a fixed pipeline layout L results in an updated KG_λ when the new data sources S are integrated in an initial knowledge graph KG_0 described by an ontology $\mathcal{O}$. The execution P can be formalized as $KG_\lambda = P(L, \lambda; S, KG_0, \mathcal{O})$. Thus, different configurations process the same source and integration context, but may produce knowledge graphs of different quality.

3.1. Budget-Constrained Quality Optimization

The goal of our approach is to identify a pipeline configuration λ so that the execution results in a high-quality KG_λ. Therefore, we consider the following complementary quality dimensions described by the KGI-Bench benchmark [27]: correctness $q_{\text{cor}}(KG_\lambda)$, coverage $q_{\text{cov}}(KG_\lambda)$, and consistency $q_{\text{con}}(KG_\lambda)$. Coverage measures how much of the expected reference content is represented in the constructed graph, considering both entities and triples while excluding information already present in the seed KG. Correctness measures how much of the produced content corresponds to the reference graph, using label-based entity alignment and normalized literal comparison; triples receive credit

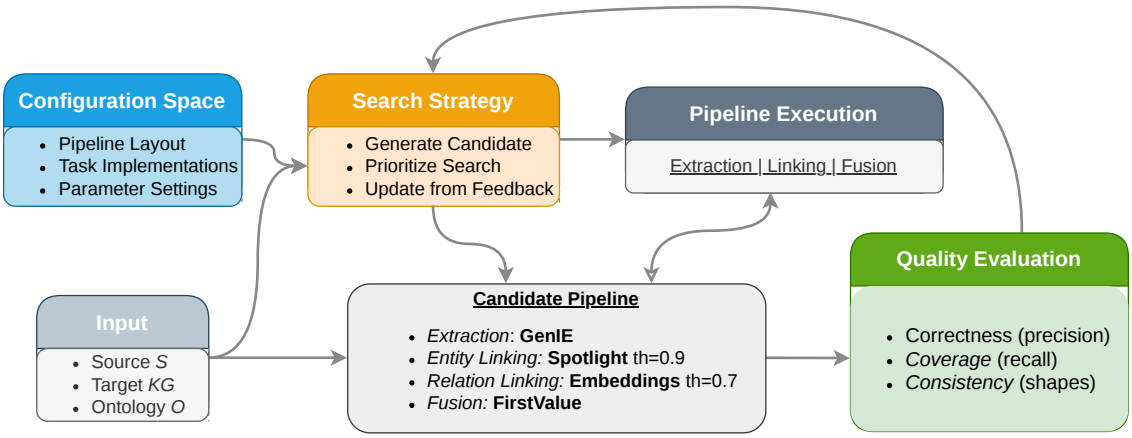


Figure 1: Overview of the quality-aware configuration search approach. The search strategy selects a valid pipeline configuration from the configuration space, instantiates and executes it on the input data, and uses the resulting knowledge graph quality as feedback for subsequent configuration choices. When the budget (B) is reached, it returns the best found pipeline of the search strategy.

only when subject, predicate, and object match, while duplicate entities reduce entity correctness. Consistency is assessed independently of the reference graph and captures compliance with ontology constraints concerning class disjointness, relation domains and ranges, relation direction, datatypes, and literal formats. In our configuration search, we aggregate these dimensions into a scalar objective $Q(\lambda)$ by calculating the harmonic mean. Without computational constraints, the optimal configuration $\lambda^* = \arg\max_{\lambda \in \Lambda_L} Q(\lambda)$, which maximizes quality over all valid configurations in Λ_L.

However, evaluating $Q(\lambda)$ requires instantiating and executing the complete pipeline and assessing the resulting graph. Due to the computationally intensive evaluation, only a limited number of configurations B can be evaluated in practice.

Let $H_t = \{(\lambda_1, Q(\lambda_1)), \ldots, (\lambda_t, Q(\lambda_t))\}$ denote the history of observations available after t pipeline executions. A search strategy uses H_t to select the next configuration λ_{t+1}. After exhausting the evaluation budget, it returns the best observed configuration $\hat{\lambda}_B = \arg\max_{\lambda_i:(\lambda_i, Q(\lambda_i)) \in H_B} Q(\lambda_i)$. The objective of quality-aware pipeline search is therefore to obtain a configuration $\hat{\lambda}_B$ whose quality is as close as possible to the global optimum while satisfying $|H_B| \leq B$.

4. Pipeline Search

We now present the search framework used to solve the budget-constrained configuration problem introduced in Section 3.1. As illustrated in Figure 1, the framework follows an iterative generate–execute–evaluate–update loop. A search strategy selects a candidate configuration, executes the instantiated pipeline with a suitable tool such as KGpipe, and evaluates the quality of the resulting knowledge graph using it as feedback for subsequent configuration choices.

4.1. Framework Overview

The framework separates configuration search from pipeline execution and quality evaluation. A search strategy operates over the valid configuration space Λ_L defined in Section 3.1. Based on the observations H_t collected during the first t evaluations, it selects a configuration λ_{t+1} for evaluation.

KGpipe instantiates the fixed pipeline layout L according to λ_{t+1}, executes the resulting pipeline on the integration scenario, and produces the knowledge graph $KG_{\lambda_{t+1}}$. The graph is assessed using the quality objective Q, and the resulting observation $\big(\lambda_{t+1}, Q(\lambda_{t+1})\big)$ is added to the search history. The updated history is then available to the search strategy when selecting the next configuration.

This separation allows different search strategies to operate on the same configuration space and under the same evaluation protocol. The search strategy does not require knowledge of the internal execution logic of the individual pipeline components; it only proposes configurations and receives their end-to-end quality scores. Because the feedback is computed from the resulting knowledge graph, it captures interactions between decisions made across different pipeline tasks.

The procedure continues until the evaluation budget B is exhausted and returns the best observed configuration $\hat{\lambda}_B$, as defined in Section 3.1.

4.2. Candidate Generation

The candidate generator provides the interface between a search strategy and the executable KGpipe pipeline. Each generated candidate is an element $\lambda \in \Lambda_L$ and therefore specifies one implementation and its associated parameter values for every task in the fixed pipeline layout.

For a task T_j, the generator selects an implementation $i_j \in \mathcal{I}_j$ and assigns values only to the parameters defined by the corresponding domain Θ_{j,i_j}. Parameters that do not apply to the selected implementation are omitted. Candidate generation must therefore account for the conditional structure of the configuration space in addition to categorical and numeric parameter choices.

Before execution, KGpipe verifies that the selected implementations and parameter values form a valid pipeline configuration. In particular, all parameter values must lie within their implementation-specific domains, and the selected components must be compatible with the input and output requirements of the fixed layout. Only valid configurations are passed to pipeline execution. Since the layout is fixed, candidate generation does not add, remove, or reorder pipeline tasks.

4.3. Search Strategies

We compare five search-strategy families for exploring the valid configuration space: uniform random search, implementation-aware random search, Random Neighborhood Search, Heuristic Neighborhood Refinement, and Bayesian optimization. HNR is evaluated in two variants, HNR-1 and HNR-2. All strategies operate under the same evaluation budget B and return the best observed configuration $\hat{\lambda}_B$.

After t evaluations, the search history is $H_t = \{(\lambda_1, Q(\lambda_1)), \ldots, (\lambda_t, Q(\lambda_t))\}$. A strategy selects an unevaluated configuration $\lambda_{t+1} \in \Lambda_L \setminus \{\lambda_1, \ldots, \lambda_t\}$ for the next pipeline execution. The strategies differ in how they use the observed quality scores in H_t.

The compared strategies represent increasing use of structure and feedback. Random search does not use observed quality scores. RNS uses quality feedback to refine high-performing configurations through generic local modifications. HNR additionally exploits the task structure of KG pipelines by separating implementation-level and parameter-level refinement. Bayesian optimization uses quality feedback to fit a global surrogate model of the objective function.

4.3.1. Random Search

Random search serves as a strategy-independent baseline. At each iteration, it samples a previously unevaluated valid pipeline configuration from the configuration space and executes it. The observed quality scores are not used to guide future selections, making each evaluation independent of previous search results. Random search, therefore, provides a reference for assessing whether exploiting quality feedback improves configuration search under the same evaluation budget.

4.3.2. Implementation-aware Search

A second random strategy exploits the hierarchical structure of the configuration space. A pipeline configuration consists of two types of decisions: implementation choices, which determine the coarse structure of the instantiated pipeline, and parameter bindings, which refine the behavior of the selected implementations.

The initialization enumerates all valid combinations of task implementations whenever their number is sufficiently small. For each implementation combination, multiple parameter bindings are sampled, yielding a set that covers the implementation space while introducing variation in the associated parameter settings. If the number of implementation combinations exceeds the available budget, a random subset of implementation combinations is selected before generating parameter bindings.

This strategy is motivated by the observation that KG pipeline search spaces are often dominated by parameter combinations, while the number of available implementations per task remains comparatively small. Covering the implementation space early allows the subsequent search phase to refine promising pipeline structures rather than discovering them only by chance.

4.3.3. Neighborhood Search

Random Neighborhood Search (RNS) is a generic local-search strategy. It exploits the assumption that promising configurations may have promising local variants. The method maintains a set of high-quality anchor configurations and explores one-step modifications around them, while occasionally performing global random exploration. For a configuration λ we define its neighborhood as the set of valid configurations that differ from λ in exactly one implementation choice or one active parameter value. A neighbor differs from λ in exactly one of the following ways.

Parameter modification. Parameter modification performs fine-grained refinement within the currently selected pipeline structure. One active parameter of one selected implementation is changed, while all implementation choices and all other parameter values remain fixed. For ordered parameter domains, nearby admissible values are considered first, allowing the search to move gradually toward better settings. For categorical domains (finite set of named alternatives, e.g., model selection), the current value is replaced by another valid category. Previously evaluated variants are excluded.

Implementation replacement. The implementation of exactly one task T_j is replaced by another implementation from $\mathcal{I}_j$, while all other task implementations remain fixed. Parameters that no longer apply are removed. Parameters of the newly selected implementation are initialized using their default value if available; otherwise, they are sampled from the corresponding domain.

At iteration t, let $\mathrm{TopK}(H_t, k)$ denote the set of the k configurations with the highest observed quality scores. With probability ρ, RNS performs a global exploration step by sampling an unevaluated configuration from Λ_L. With probability $1-\rho$, it samples an anchor configuration from $\mathrm{TopK}(H_t, k)$ and evaluates an unevaluated neighbor of that anchor. If the selected anchor has no remaining unevaluated neighbors, another anchor is tried. If no anchor in $\mathrm{TopK}(H_t, k)$ has an unevaluated neighbor, the method falls back to global random exploration.

The beam size k controls how many high-quality configurations are used as search anchors. The exploration rate ρ controls the balance between local refinement and global exploration.

4.3.4. Heuristic Neighborhood Refinement

The default neighborhood search treats implementation and parameter changes uniformly. However, KG pipeline configurations have a hierarchical structure: implementation choices determine the coarse behavior of the pipeline, whereas parameter bindings refine the selected implementations. Heuristic Neighborhood Refinement (HNR) exploits this structure by performing task-local refinement and prioritizing implementation changes before parameter changes.

For a configuration λ, let $\mathcal{N}^{(j)}_{\mathrm{impl}}(\lambda)$ denote the set of valid configurations obtained by replacing the implementation of task T_j, and let $\mathcal{N}^{(j)}_{\mathrm{param}}(\lambda)$ denote the set obtained by modifying one active parameter of the implementation selected for T_j. All other settings remain fixed.

Starting from an evaluated initial set, HNR repeatedly refines a high-quality configuration using these task-local neighborhoods. If an evaluated candidate improves the current best configuration, it becomes the new refinement anchor. Implementation alternatives are considered before parameter modifications, because they are expected to induce larger changes in pipeline behavior. Global random exploration (ρ) can be set but is disabled for our experiments.

Table 1
Configuration spaces of the evaluated pipeline families. ST denotes similarity thresholds in $\{0.5, 0.6, \ldots, 0.9\}$, and LM denotes the language models `all-MiniLM-L6-v2`, `all-mpnet-base-v2`, and `e5-base-v2`. All pipelines end with FirstValue fusion, which prioritizes seed-KG values for matched entities and single-valued properties. (*) Combined matching jointly performs ontology and entity matching within one task.

Source	Task	Implementation with Parameters
RDF	Ontology Matching	Embedding-based [LM, ST], PARIS [ST]
RDF	Entity Matching	Embedding-based [LM, ST], PARIS [ST]
RDF	Combined Matching*	Embedding-based [LM, ST], PARIS [ST]
Text	Information Extraction	OpenIE [none], GenIE [none]
Text	Entity Linking	Embedding-based [LM, ST], Spotlight [ST]
Text	Relation Linking	Embedding-based [LM, ST], REBEL [ST]

We consider two variants that differ only in how refinement is scheduled across pipeline tasks. **HNR-1** cycles through the tasks in pipeline order and performs one refinement step for each task before restarting from the first task. **HNR-2** focuses on one task at a time and continues refining it until no meaningful improvement is observed. The search advances to the next task after r consecutive evaluations with a quality gain below a threshold $\delta_{\min}$.

Both variants modify only one task-local decision per evaluation, which makes their search behavior interpretable and enables the contribution of individual pipeline stages to be analyzed.

4.3.5. Bayesian Optimization

Bayesian optimization [19] uses the observed configurations and their quality scores to construct a model of the objective function. After an initial set of configurations has been evaluated, a surrogate model $\mathcal{M}_t$ is fitted to the search history $H_t = \{(\lambda_1, Q(\lambda_1)), \ldots, (\lambda_t, Q(\lambda_t))\}$. An acquisition function $a(\lambda; \mathcal{M}_t)$ then selects the next configuration by balancing predicted quality and model uncertainty: $\lambda_{t+1} = \arg\max_{\lambda \in \Lambda_L \setminus \{\lambda_1, \ldots, \lambda_t\}} a(\lambda; \mathcal{M}_t)$.

The configuration space considered in this work is mixed and conditional: implementation choices are categorical, parameters may be categorical or numeric, and only the parameters of the selected implementations are active. Therefore, we use a Bayesian optimization method that supports conditional configuration spaces. The concrete surrogate model, acquisition function, initialization strategy, and hyperparameter settings are reported in the experimental setup.

4.4. Quality Feedback

After executing a configuration λ, KGpipe evaluates the resulting knowledge graph KG_λ using the quality dimensions introduced in Section 3.1.

The score aggregations are fixed before the search and remain identical for all evaluated configurations and search strategies. This ensures that the observed scores are directly comparable throughout the optimization process. The quality score is computed from the final knowledge graph rather than from the outputs of individual pipeline tasks. It therefore reflects the combined effect of implementation and parameter decisions across the complete pipeline, including interactions and error propagation between tasks.

After evaluation, the observation $\big(\lambda, Q(\lambda)\big)$ is added to the search history H_t. The search strategies use this feedback differently: random searches do not consider previous scores, RNS and HNR use them to identify promising search anchors, and Bayesian optimization uses them to update its surrogate model. The concrete definitions and computation of the quality dimensions are described in [27].

5. Evaluation

Benchmark and Data. We use KGI-Bench [27] as a benchmark for evaluating KG integration pipelines. The benchmark provides heterogeneous input sources, a seed knowledge graph, a target ontology, a reference knowledge graph, and metrics for assessing end-to-end KG quality. In our experiments, we use the larger 10k variant but integrate only a single source rather than adding multiple source increments.

We evaluate two representative pipeline families. The RDF pipeline integrates structured sources and consists of ontology matching, entity matching, and data fusion. The text pipeline integrates unstructured sources and consists of information extraction, entity linking, relation linking, and data fusion. This allows us to study configuration search for both structured and text-based KG integration scenarios.

Configuration Space. Table 1 summarizes the configuration spaces used in the experiments. For each pipeline family, the search space contains categorical implementation choices and task-specific parameter bindings. The resulting configuration spaces are mixed and conditional, since the active parameters depend on the selected implementation.

The total number of valid configurations is 440 for the RDF pipeline and 600 for the text pipeline. All methods operate on the same valid configuration space and are evaluated under the same budget.

Evaluation Protocol. Each search method is run with a fixed evaluation budget of $B = 20$ pipeline executions. The same budget is used for all methods and both pipeline families. We repeat each experiment with ten random seeds and report the mean and standard deviation across runs.

For RNS the probability of global random exploration is set to $\rho = 0.2$. RNS retains the top $k = 3$ evaluated configurations as candidate anchors. HNR-2 advances to the next pipeline task after $r = 3$ consecutive evaluations without a meaningful improvement, where an improvement is considered meaningful if the quality gain exceeds $\delta_{\min} = 0.01$. Bayesian optimization uses a similarity-based surrogate model with a UCB acquisition function and is initialized with one implementation-aware evaluated configuration.

Quality Metrics. The quality of a configuration is measured using the scalar objective introduced in Section 3.1. We report five types of performance measures. First, Q_{best} denotes the best quality found within the budget, summarized by its mean and standard deviation across runs. Second, *Hits max* reports the number of runs that recover the global maximum identified by exhaustive grid search evaluation. Third, *Evals to 95%* and *100%* report the mean number of executions required to reach the corresponding fraction of the highest score found by the specific optimization approach. Fourth, *AUC* summarizes best-so-far quality across the complete search trajectory, with higher values indicating better sample efficiency.

6. Results

We evaluate whether quality-aware configuration search can identify high-quality KG integration pipelines under a limited evaluation budget.

The exhaustive evaluation showed score ranges of 0.4239–0.7712 for RDF and 0.2614–0.3803 for text. The RDF space contains several configurations near the optimum, whereas the text scores are concentrated in a comparatively narrow range. The complete score distributions and exhaustive-search results are available in the accompanying repository.

Table 2 summarizes the search performance under the fixed evaluation budget, while Figures 2 and 3 show the best-so-far quality over the evaluated configurations.

RDF pipelines. For the RDF pipeline, Bayesian optimization achieves the highest mean best quality, closely followed by random search. Implementation-aware search, however, reaches high-quality configurations particularly early and records the highest *Hits max*, reaching the global maximum in 5 of 10 runs.

This performance is partly explained by its non-uniform sampling distribution. The method first

Table 2
Overall search performance under evaluation budget B. Mean best quality and standard deviation are computed over repeated runs. "95%/100%" denotes the mean number of evaluations required to reach 95%/100% of the respective search method (local, not global maximum). Search AUC summarizes sample efficiency (higher is better). The grid search found RDF pipeline scores in [0.4239-0.7712] and for text in [0.2614-0.3803].

	Strategy	Q best (mean)	Hits max	Evals to 95%	Evals to 100%	AUC
RDF	Random	0.7667 ± 0.0084	0/10	6.40 ± 3.41	11.30 ± 6.31	13.89 ± 0.33
	Implementation-aware	0.7603 ± 0.0208	**5/10**	**3.60** ± 3.41	**8.50** ± 5.64	**14.11** ± 0.49
	RNS	0.7530 ± 0.0265	1/10	9.20 ± 6.09	14.60 ± 3.66	13.33 ± 0.87
	HNR-1	0.7636 ± 0.0121	1/10	8.00 ± 5.06	10.80 ± 5.67	13.79 ± 0.45
	HNR-2	0.7656 ± 0.0113	2/10	8.80 ± 3.99	11.70 ± 5.38	13.62 ± 0.66
	Bayesian optimization	**0.7687** ± 0.0064	1/10	8.10 ± 5.22	12.30 ± 5.79	13.79 ± 0.45
TEXT	Random	0.3739 ± 0.0062	0/10	6.70 ± 3.23	**9.00** ± 3.80	6.90 ± 0.17
	Implementation-aware	0.3763 ± 0.0015	0/10	5.80 ± 4.18	10.40 ± 6.08	6.91 ± 0.21
	RNS	0.3710 ± 0.0152	**2/10**	6.20 ± 6.37	12.00 ± 4.69	6.79 ± 0.35
	HNR-1	0.3751 ± 0.0077	1/10	**3.40** ± 1.96	12.60 ± 5.72	**7.00** ± 0.15
	HNR-2	**0.3772** ± 0.0022	1/10	6.40 ± 4.67	12.00 ± 5.52	6.90 ± 0.21
	Bayesian optimization	0.3762 ± 0.0028	1/10	6.50 ± 5.84	11.00 ± 4.71	6.90 ± 0.26

samples an implementation assignment and then its parameter values, giving concrete configurations from smaller parameter spaces a higher sampling probability. Since the RDF optimum lies in one of these oversampled regions, implementation-aware search is more likely than uniform random search to evaluate it. Its strong RDF performance therefore reflects both broad implementation coverage and the location of the optimum within the hierarchical search space. Random search nevertheless performs competitively on RDF. The distribution of RDF scores indicates that several configurations achieve scores close to the maximum. Uniform exploration can therefore find strong configurations without requiring a precise local refinement path. Bayesian optimization benefits from the same relatively regular search landscape and achieves the highest average final score, although it does not consistently recover the exact maximum within the limited budget.

RNS remains the least consistent local-search method on the RDF pipeline, showing both the lowest mean best quality and comparatively high variance. The two HNR variants perform more competitively. In particular, HNR-2 achieves a mean best quality close to random search and Bayesian optimization, while HNR-1 also improves over RNS. This suggests that separating implementation-level and parameter-level refinement provides a more useful local-search structure than treating all neighborhood modifications uniformly. HNR-2 performs slightly better than HNR-1, indicating that concentrating multiple evaluations on one task can be more effective than switching tasks after every refinement step.

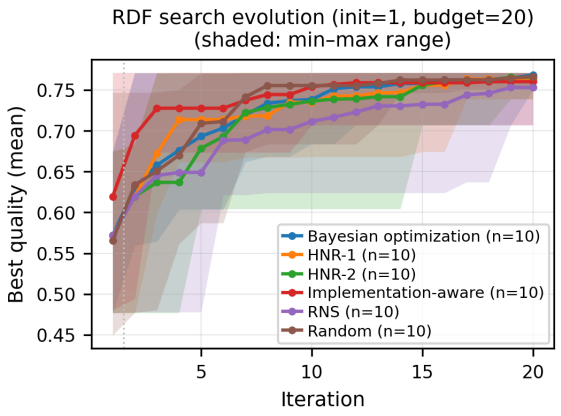


Figure 2: RDF pipeline search, best scores.

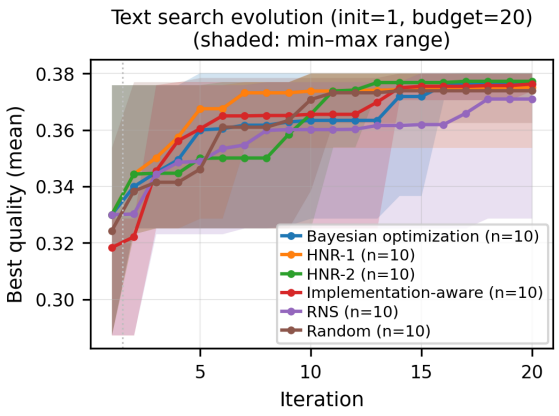


Figure 3: Text pipeline search, best scores.

Nevertheless, neither HNR variant consistently outperforms the global search strategies on RDF.

Text pipelines. For the text pipeline, the differences between the methods remain small. HNR-2 achieves the highest mean best quality, while HNR-1 attains the highest search AUC and reaches 95% of the best observed quality with the fewest evaluations. Implementation-aware search and Bayesian optimization obtain nearly identical final scores. The text pipelines have a narrow score range, as shown in Figure 3, suggesting that many configurations produce similar end-to-end quality. Consequently, the choice of search strategy has less influence on the final result than for the RDF pipeline.

HNR-1 rises comparatively quickly in the convergence plot, indicating that sequential task-local refinement can exploit small improvements in the text pipeline. This may be because the text pipeline contains several stages whose parameter and implementation choices contribute incrementally to the final quality. Repeatedly revisiting the tasks allows HNR-1 to accumulate such local gains across the pipeline.

HNR-2 shows a different trade-off. Its task-focused refinement does not reach high-quality configurations as early as HNR-1, but it achieves the highest mean best quality at the end of the evaluation budget. This suggests that remaining on one task for several evaluations can support more thorough local refinement, although the resulting gains may appear later in the search. The difference between the two variants indicates that sequentially revisiting tasks is beneficial for early progress, whereas sustained task-wise refinement can be advantageous for final quality.

RNS also shows comparatively high variance. Its generic neighborhood treats all local modifications uniformly and does not distinguish between implementation-level and parameter-level effects. In a pipeline with several heterogeneous stages, randomly selected local changes may often have little effect or may modify a task that is not currently limiting overall quality.

Bayesian optimization remains competitive but does not clearly outperform the simpler strategies. The text search space is mixed and conditional, while the evaluation budget is small relative to the number of configurations. Under these conditions, the surrogate model receives only limited evidence for estimating interactions between task implementations and active parameters. The narrow variation in observed quality further reduces the signal available to distinguish promising regions.

Overall comparison and limitations. No single strategy dominates both pipeline families. On the RDF pipeline, Bayesian optimization achieves the highest mean best quality, while random search and HNR-2 follow closely. Implementation-aware search is particularly effective at discovering the RDF maximum and reaching high-quality configurations early, partly because of its non-uniform hierarchical sampling. On the text pipeline, HNR-1 provides the best early sample efficiency, whereas HNR-2 achieves the highest mean best quality after the full budget. Nevertheless, the differences between the text results are small. Overall, search effectiveness depends strongly on the structure of the configuration space, the distribution of high-quality configurations, and the degree to which local modifications yield predictable quality improvements.

From a practical perspective, our results suggest that simple global strategies remain strong choices when high-quality configurations are relatively dense or quality differences across the search space are small. Implementation-aware sampling can be beneficial when implementation choices structure the search space strongly and broad implementation coverage is desirable. Structure-aware neighborhood refinement appears more useful when task-local changes yield gradual and predictable quality improvements. These observations should be interpreted as guidance for the studied scenarios rather than as general strategy-selection rules.

The evaluation is limited to two pipeline families, one integration scenario per source type, and relatively small enumerable configuration spaces. Accordingly, the results characterize the behavior of the compared strategies in these scenarios rather than establish a universally superior search method.

7. Conclusion

We formulated the configuration of KG integration pipelines as a budget-constrained optimization problem over task implementations and parameters. Our pipeline search framework iteratively generates,

executes, and evaluates valid pipeline configurations and supports different search strategies under a common evaluation protocol.

The evaluation on RDF and text-based pipelines shows that strong configurations can be found after evaluating only a small part of the search space. At the same time, no strategy consistently dominates: Bayesian optimization, implementation-aware sampling, and heuristic neighborhood refinement provide advantages for different pipeline structures. This highlights the importance of matching the search strategy to the structure and quality landscape of the configuration space.

Future work will study adaptive selection between implementation-level and parameter-level refinement, dynamic allocation of the initialization and search budgets, and cost-aware objectives that jointly consider KG quality and pipeline runtime. We also plan to support supervised pipeline tasks using active learning and LLM-based feedback. To further enhance flexibility, we will investigate non-fixed pipeline layouts, expanding the search space to identify qualitative configurations. This approach will enable more robust and scalable solutions for diverse KG integration tasks.

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.

Declaration of use of Generative AI During the preparation of this work, the authors used OpenAI ChatGPT for grammar and spelling checks, paraphrasing and rewording, and improving the writing style of the manuscript. Cursor was used to assist with the generation of boilerplate and interface code and with the development of tests based on the authors' specifications and the existing codebase. All AI-assisted text and code were reviewed, edited, tested, and revised by the authors as appropriate. The authors take full responsibility for the implementation and the content of the publication.

References

[1] M. Hofer, E. Rahm, Kgpipe: Generation of pipelines for data integration into knowledge graphs, in: 15th International Workshop on Quality in Databases (QDB'26), VLDB 2026, 2026. URL: https://vldb.org/2026/Workshops/VLDB-Workshops-2026/QDB/QDB26_7.pdf.

[2] M. Hofer, D. Obraczka, A. Saeedi, H. Köpcke, E. Rahm, Construction of knowledge graphs: Current state and challenges, Information 15 (2024) 509.

[3] C. Niklaus, M. Cetto, A. Freitas, S. Handschuh, A survey on open information extraction, in: Proceedings of the 27th International Conference on Computational Linguistics, Association for Computational Linguistics, Santa Fe, New Mexico, USA, 2018, pp. 3866–3878. URL: https://aclanthology.org/C18-1326/.

[4] I. L. Oliveira, R. Fileto, R. Speck, L. P. Garcia, D. Moussallem, J. Lehmann, Towards holistic entity linking: Survey and directions, Information Systems 95 (2021) 101624. URL: https://www.sciencedirect.com/science/article/pii/S0306437920300958. doi:https://doi.org/10.1016/j.is.2020.101624.

[5] R. Liu, R. Mao, A. Luu, E. Cambria, A brief survey on recent advances in coreference resolution, Artificial Intelligence Review 56 (2023) 14439–14481. URL: https://doi.org/10.1007/s10462-023-10506-3. doi:10.1007/s10462-023-10506-3.

[6] E. Jiménez-Ruiz, B. C. Grau, Y. Zhou, Logmap 2.0: towards logic-based, scalable and interactive ontology matching, in: A. Paschke, A. Burger, P. Romano, M. S. Marshall, A. Splendiani (Eds.), Proceedings of the 4th International Workshop on Semantic Web Applications and Tools for the Life Sciences, SWAT4LS 2011, London, United Kingdom, December 07-09, 2011, ACM, 2011, pp. 45–46. URL: https://doi.org/10.1145/2166896.2166911. doi:10.1145/2166896.2166911.

[7] Y. Li, J. Li, Y. Suhara, A. Doan, W. Tan, Deep entity matching with pre-trained language models,

Proc. VLDB Endow. 14 (2020) 50–60. URL: http://www.vldb.org/pvldb/vol14/p50-li.pdf. doi:10.14778/3421424.3421431.

[8] A. Doan, P. Konda, P. S. G. C., Y. Govind, D. Paulsen, K. Chandrasekhar, P. Martinkus, M. Christie, Magellan: toward building ecosystems of entity matching solutions, Commun. ACM 63 (2020) 83–91. URL: https://doi.org/10.1145/3405476. doi:10.1145/3405476.

[9] A. N. Ngomo, M. A. Sherif, K. Georgala, M. M. Hassan, K. Dreßler, K. Lyko, D. Obraczka, T. Soru, LIMES: A framework for link discovery on the semantic web, Künstliche Intell. 35 (2021) 413–423. URL: https://doi.org/10.1007/s13218-021-00713-x. doi:10.1007/S13218-021-00713-X.

[10] F. Castanedo, A review of data fusion techniques, The Scientific World Journal 2013 (2013) 704504. URL: https://onlinelibrary.wiley.com/doi/abs/10.1155/2013/704504. doi:https://doi.org/10.1155/2013/704504. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1155/2013/704504.

[11] F. M. Suchanek, S. Abiteboul, P. Senellart, Paris: probabilistic alignment of relations, instances, and schema, Proc. VLDB Endow. 5 (2011) 157–168. URL: https://doi.org/10.14778/2078331.2078332. doi:10.14778/2078331.2078332.

[12] S. Lacoste-Julien, K. Palla, A. Davies, G. Kasneci, T. Graepel, Z. Ghahramani, Sigma: simple greedy matching for aligning large knowledge bases, in: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '13, Association for Computing Machinery, New York, NY, USA, 2013, p. 572–580. URL: https://doi.org/10.1145/2487575.2487592. doi:10.1145/2487575.2487592.

[13] H. Bian, Llm-empowered knowledge graph construction: A survey, CoRR abs/2510.20345 (2025). URL: https://doi.org/10.48550/arXiv.2510.20345. doi:10.48550/ARXIV.2510.20345. arXiv:2510.20345.

[14] M. Y. Jaradeh, K. Singh, M. Stocker, A. Both, S. Auer, Better call the plumber: orchestrating dynamic information extraction pipelines, in: International Conference on Web Engineering, Springer, 2021, pp. 240–254.

[15] K. Nikoletos, V. Efthymiou, G. Papadakis, K. Stefanidis, Auto-configuring entity resolution pipelines, IEEE Access (2025).

[16] A. Steiner, C. Bizer, Automatic end-to-end data integration using large language models, arXiv preprint arXiv:2603.10547 (2026).

[17] V. Christen, D. Obraczka, M. Hofer, M. Franke, E. Rahm, Graph metrics-driven record cluster repair meets llm-based active learning, ACM J. Data Inf. Qual. 17 (2025) 7:1–7:25. URL: https://doi.org/10.1145/3735511. doi:10.1145/3735511.

[18] F. Hutter, H. H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers, volume 6683 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 507–523. URL: https://doi.org/10.1007/978-3-642-25566-3_40. doi:10.1007/978-3-642-25566-3_40.

[19] J. Snoek, H. Larochelle, R. P. Adams, Practical bayesian optimization of machine learning algorithms, in: Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States, 2012, pp. 2960–2968. URL: https://proceedings.neurips.cc/paper/2012/hash/05311655a15b75fab86956663e1819cd-Abstract.html.

[20] P. Koch, O. Golovidov, S. Gardner, B. Wujek, J. Griffin, Y. Xu, Autotune: A derivative-free optimization framework for hyperparameter tuning, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '18, Association for Computing Machinery, New York, NY, USA, 2018, p. 443–452. URL: https://doi.org/10.1145/3219819.3219837. doi:10.1145/3219819.3219837.

[21] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A next-generation hyperparameter optimization framework, in: A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, G. Karypis (Eds.), Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019, ACM, 2019, pp. 2623–2631. URL:

https://doi.org/10.1145/3292500.3330701. doi:10.1145/3292500.3330701.
[22] H. Paulheim, Knowledge graph refinement: A survey of approaches and evaluation methods, Semantic Web 8 (2017) 489–508. URL: https://doi.org/10.3233/SW-160218. doi:10.3233/SW-160218.
[23] S. Marchesin, G. Silvello, Efficient and reliable estimation of knowledge graph accuracy, Proc. VLDB Endow. 17 (2024) 2392–2404. URL: https://www.vldb.org/pvldb/vol17/p2392-marchesin.pdf. doi:10.14778/3665844.3665865.
[24] C. Cortes, L. Ehrlinger, L. Etcheverry, F. Naumann, Is SHACL suitable for data quality assessment?, CoRR abs/2507.22305 (2025). URL: https://doi.org/10.48550/arXiv.2507.22305. doi:10.48550/ARXIV.2507.22305. arXiv:2507.22305.
[25] F. Shami, S. Marchesin, G. Silvello, Benchmarking large language models for knowledge graph validation, in: Proceedings 29th International Conference on Extending Database Technology, EDBT 2026, Tampere, Finland, March 24-27, 2026, OpenProceedings.org, 2026, pp. 551–565. URL: https://doi.org/10.48786/edbt.2026.45. doi:10.48786/EDBT.2026.45.
[26] Y. Zhang, G. Xiao, A novel customizing knowledge graph evaluation method for incorporating user needs, Scientific Reports 14 (2024) 9594. URL: https://doi.org/10.1038/s41598-024-60004-x. doi:10.1038/s41598-024-60004-x.
[27] M. Hofer, E. Rahm, Evaluation of pipelines for data integration into knowledge graphs, 2026. URL: https://arxiv.org/abs/2605.22304. arXiv:2605.22304.