

FlowPipe: LLM-Enhanced Conditional Generative Flow Networks for Data Preparation Pipeline Construction

KUNYU NI, Ocean University of China, China

LEI CAO, University of Arizona, USA

JIE HE, University of Science and Technology Beijing, China

XIAOTONG ZHANG, University of Science and Technology Beijing, China

JIANFENG JIN, Northeastern University, China

JUNYU DONG, Ocean University of China, China

YANWEI YU*, Ocean University of China, China

Data preparation pipelines are a primary mechanism for improving data quality in machine learning workflows, transforming raw, error-prone tables into learning-ready data via sequential cleaning and feature transformation operators. However, automated pipeline construction remains computationally prohibitive due to the combinatorial complexity of operator sequences and the high cost of end-to-end evaluation. While Reinforcement Learning provides a principled discrete search paradigm, state-of-the-art (SOTA) Multi-DQN architectures suffer from three fundamental limitations: *structural dissonance*, where decoupled value estimators hinder long-horizon credit assignment; *semantic detachment*, where dataset context is treated as a superficial additive bias rather than strictly conditioning the agent's reasoning; and *exploration inefficiency* in a vast, sparse optimization landscape with many invalid pipeline states. To address these challenges, we propose **FlowPipe**, a unified framework that reformulates pipeline synthesis as conditional probabilistic flow generation over a directed acyclic graph. First, FlowPipe employs a Conditional Generative Flow Networks (C-GFlowNets) optimized via a Trajectory Balance objective, establishing a direct gradient path from terminal validation rewards to early actions to ensure holistic credit assignment. Second, to resolve semantic detachment, we introduce *Deep Semantic Modulation* via Feature-wise Linear Modulation (FiLM), which allows LLM-derived logical priors to multiplicatively modulate the policy's internal activation maps, thereby structurally adapting the decision logic to the dataset context. Finally, to mitigate exploration inefficiency, we incorporate failure awareness into the flow objective to prune semantically invalid states early and concentrate search mass on high-potential regions. Extensive experiments on two benchmark suites comprising 74 real-world datasets show that FlowPipe significantly outperforms SOTA baselines, improving accuracy by an average of **11.96%** while achieving a **12.5×** speedup in training convergence. The source code is available at <https://github.com/KunyuNi/FlowPipe>.

CCS Concepts: • **Information systems** → **Data analytics**; **Data cleaning**; • **Computing methodologies** → **Machine learning**.

Additional Key Words and Phrases: Data Analytics Pipeline; Data Preparation; Reinforcement learning

*Corresponding author.

Authors' Contact Information: Kunyu Ni, Ocean University of China, China, kyni@stu.ouc.edu.cn; Lei Cao, University of Arizona, USA, lcao@csail.mit.edu; Jie He, University of Science and Technology Beijing, China, hejie@ustb.edu.cn; Xiaotong Zhang, University of Science and Technology Beijing, China, xzt@ustb.edu.cn; Jianfeng Jin, Northeastern University, China, jinjf@atm.neu.edu.cn; Junyu Dong, Ocean University of China, China, dongjunyu@ouc.edu.cn; Yanwei Yu, Ocean University of China, China, yuyanwei@ouc.edu.cn.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2026 Copyright held by the owner/author(s).

ACM 2836-6573/2026/9-ART274

<https://doi.org/10.1145/3837112>

ACM Reference Format:

Kunyu Ni, Lei Cao, Jie He, Xiaotong Zhang, Jianfeng Jin, Junyu Dong, and Yanwei Yu. 2026. FlowPipe: LLM-Enhanced Conditional Generative Flow Networks for Data Preparation Pipeline Construction. *Proc. ACM Manag. Data* 4, 4, Article 274 (September 2026), 27 pages. <https://doi.org/10.1145/3837112>

1 Introduction

Data quality has emerged as the decisive factor in the lifecycle of machine learning (ML) systems. In real applications, the performance of these systems is largely bounded by the veracity and integrity of the underlying data [18, 48]. Turning raw, noisy datasets into high quality, ML ready data requires rigorous *data preparation pipelines* that orchestrate heterogeneous operators to cleanse artifacts, impute missing values, and rectify distributional anomalies [1, 31].

Designing an effective data preparation pipeline is challenging due to the vast combinatorial space of available cleaning and transformation operators. So far it remains a labor-intensive process that relies on human expertise [43, 46, 60]. For example, domain experts utilize rich *semantic context*—such as attribute definitions, domain constraints, and valid value ranges—to orchestrate operator selection, ensuring that cleaning decisions align with the intrinsic logic of the data [10].

If there were a system that could automate this process, it would be to the great need of numerous ML-driven applications. However, this is challenging. The optimization landscape is combinatorial and non-convex, and naively applying operators without semantic awareness often fails to resolve underlying quality issues or, worse, introduces new artifacts (e.g., invalid imputations) [40, 51]. The complexity is further exacerbated by the high computational burden of evaluation, which depends on delayed supervision signals from downstream models rather than explicit ground truth. Consequently, practitioners frequently default to generic configurations or inefficient trial-and-error strategies [15, 29], leaving critical data quality problems unresolved.

Recent work has extended Automatic Machine Learning (AutoML) beyond hyperparameter tuning and architecture search to *automated data preparation pipeline construction* [5, 15, 44, 56]. Existing systems broadly follow two paradigms: *differentiable pipelines* and *discrete construction*. Differentiable approaches [13, 33, 35] relax operator selection into continuous parameters to enable gradient-based optimization. However, this method is inherently constrained by the non-differentiable nature of many essential cleaning operators and incurs prohibitive computational overhead due to repetitive end-to-end gradient updates. On the other hand, discrete search methods, such as evolutionary algorithms TPOT [40] and Bayesian optimization Auto-Weka [51], treat pipeline evaluation as a monolithic black-box problem. These approaches often struggle to capture the fine-grained, state-dependent transitions required to assemble complex, multi-step workflows.

Consequently, Reinforcement Learning (RL) has emerged as the premier framework within the discrete paradigm. By formulating pipeline construction as a sequential decision process, RL naturally handles the discrete operators that differentiable methods cannot. Moreover, it explicitly captures the step-wise dependencies that monolithic evolutionary search algorithms often overlook. This approach perfectly aligns with data preparation tasks, where the utility of an operator is dictated by the context created by preceding transformations.

However, translating this theoretical alignment into a robust system remains a significant challenge. Pioneer RL-based works, such as DeepLine [20], modeled pipeline generation using a unified DQN agent. While this monolithic design simplifies the architecture, it struggles within the high-dimensional action space, where the combinatorial complexity of operators and parameters grows exponentially. To address this, state-of-the-art (SOTA) systems like HaiPipe [10] and CtxPipe [16] adopt a Multi-DQN architecture, as shown in Figure 1(a). This approach isolates the decision logic for each component of the pipeline, such as Imputation, Encoding, and Feature Engineering, into

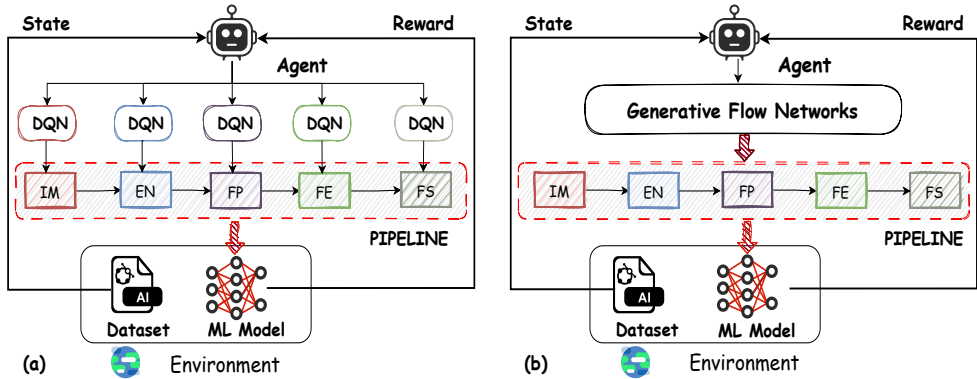


Fig. 1. **Paradigm Comparison.** (a) **The Multi-DQN architecture** employs decoupled agents for each stage, causing structural dissonance and credit assignment problems. (b) **FlowPipe** models the entire pipeline synthesis as a coherent probabilistic flow via GFlowNets, ensuring holistic credit assignment. The target data preparation pipeline consists of five sequential stages: Imputer (IM), Encoder (EN), Feature Preprocessing (FP), Feature Engineering (FE), and Feature Selection (FS).

separate networks. Each DQN is tasked with the decision-making for a specific pipeline component, facilitating more efficient management of the vast search space. **Challenges.** Despite these advances, existing solutions face three fundamental obstacles that limit their effectiveness in real-world scenarios:

- **Credit Assignment Problem.** The core challenge lies in distinguishing the specific contribution of individual upstream actions to the final downstream performance. While the Multi-DQN formulation effectively decomposes the combinatorial action space into manageable sub-spaces, this design creates a fundamental *structural dissonance*. Although the operational sequence remains sequentially interdependent, the decision-making is distributed across isolated Q-networks. This separation makes it difficult to consistently attribute delayed validation rewards to upstream pipeline decisions.

- **Semantic Detachment.** Beyond the structural separation, the architecture is further limited by *Semantic Detachment*, where the dataset's semantic context fails to actively inform the agent's core reasoning process. While CtxPipe [16] pioneers the inclusion of dataset context, it mechanically injects this information as an *extrinsic bias term* added to the final value estimates. This integration strategy causes the semantic signals to bypass the agent's core reasoning machinery. Operationally, the agent evaluates the pipeline state using a fixed, generic logic while the context serves merely as an independent value adjustment applied at the output layer. Consequently, the policy remains structurally unconditioned by the dataset profile and fails to adapt its fundamental cleaning strategy to the specific semantic context of the target dataset.

- **Exploration Inefficiency.** This refers to the prohibitive computational cost of navigating an optimization landscape where the search space grows exponentially with the sequence length. Current approaches exacerbate this challenge by relying solely on shallow features like table schema or statistical metadata to guide exploration. Lacking richer semantic cues to prune this vast combinatorial manifold, and given that validation requires expensive end-to-end training, the agent is forced into a brute-force trial-and-error loop. Consequently, standard strategies suffer from severe sample inefficiency, often failing to locate optimal regions within a reasonable time budget.

Proposed Approach. To bridge the gap between RL's theoretical promise and its practical implementation, we propose FlowPipe, a system that formulates pipeline synthesis as a unified conditional probabilistic flow generation process over a linear compositional space. First, to resolve the *Credit Assignment Problem*, FlowPipe departs from the fragmented Multi-DQN architecture

by deploying a C-GFlowNets optimized via the Trajectory Balance objective. Unlike short-sighted step-wise updates that isolate decisions, our formulation explicitly models the probability flow of the complete pipeline trajectory. This establishes a direct gradient path from the terminal validation reward back to earliest actions, ensuring holistic supervision across long horizons. Simultaneously, to resolve *Semantic Detachment*, we discard the extrinsic bias approach of prior works that restricts context integration to a superficial concatenation at the final output. Instead, we employ *Deep Semantic Modulation* via FiLM layers to condition the agent’s intermediate representations on LLM-enhanced semantic context. By functioning as a dynamic filter, this mechanism selectively amplifies relevant feature signals while dampening incompatible ones deep within the network layers. Consequently, the model implicitly re-configures its core reasoning logic, ensuring that the decision-making process is structurally adapted to the unique constraints of each target dataset. Finally, to mitigate *Exploration Inefficiency*, we implement a *Failure-aware Trajectory Balance* objective. We incorporate failure awareness into the flow objective to prune invalid pipeline states early. This effectively anchors the exploration budget, pruning the exponential combinatorial space to concentrate search mass exclusively on semantically valid, high-potential regions. Extensive experiments on 74 real-world datasets confirm FlowPipe’s dominance over SOTA baselines, delivering an 11.96% average accuracy gain with 12.5× faster training.

In summary, our key contributions are as follows:

- **Generative Flow Formulation.** We propose **FlowPipe**, the first framework to formulate data preparation as a conditional generative flow. By modeling the complete pipeline as a cohesive unit, our Trajectory Balance objective establishes a direct gradient path for precise credit attribution across long-horizon dependencies.
- **Semantic-Constrained Policy.** We introduce a mechanism that conditions pipeline generation on LLM-derived logical priors. By modulating the policy network via FiLM, we enforce strict semantic consistency between data properties and operator logic.
- **Efficiency and Effectiveness.** We conduct extensive experiments across the DiffPrep and DeepLine benchmarks. FlowPipe not only establishes a new SOTA by improving accuracy by **11.17%** and **12.17%**, respectively, but also demonstrates superior computational efficiency. Compared to the best-performed baseline (*i.e.*, CtxPipe), FlowPipe achieves inference speedups of **20.99%** on DiffPrep and **26.55%** on DeepLine.

2 Related Work

2.1 Automated Pipeline Construction

Automated data preparation pipeline construction is a fundamental problem in data management. Unlike standard hyperparameter optimization, it involves searching a combinatorial space of heterogeneous operators subject to logical dependencies [11, 28]. Prior work mainly follows two paradigms: *discrete construction*, which synthesizes pipelines via sequential decisions over an operator graph, and *differentiable construction*, which relaxes discrete choices into continuous parameters for gradient-based optimization. *FlowPipe adopts the discrete paradigm and navigates the resulting search space using generative flow modeling.*

Discrete Construction. Discrete methods frame pipeline synthesis as sequential decision-making over a finite operator set [47]. Early systems such as Auto-Weka [51] and Auto-Sklearn [15] couple Bayesian optimization with meta-learning to jointly optimize preprocessing and modeling, but often reduce data preparation to a shallow, mostly fixed stage. TPOT [40] uses genetic programming to evolve tree-structured pipelines, yet can struggle to converge in high-dimensional spaces due to limited directed guidance. Similarly, SAGA [49] advances this category by employing a specialized heuristic evolutionary search for pipeline generation. More generally, evolutionary

and Bayesian approaches tend to evaluate pipelines monolithically, making it difficult to model fine-grained, state-dependent transitions typical of multi-step cleaning.

To better capture sequential dependencies, RL-based methods have become prominent. Learn2Clean [5] and DeepLine [20] model long-horizon decision processes but often rely on rigid heuristic constraints (e.g., fixed operator ordering) for tractability. For more complex settings, HAIPipe [10] and CtxPipe [16] adopt Multi-DQN designs that decompose pipeline generation into stages handled by separate agents; however, this decoupling can hinder global credit assignment because each Q-network optimizes a local objective. In CtxPipe, despite incorporating dataset context, reliance on random snapshots and shallow additive fusion may limit the ability to capture deep semantic dependencies between data and operators. More recently, SwiftDP [30] combines MCTS with meta-learning, but its reliance on meta-features from seen datasets limits generalization. ShapleyPipe [8] uses Shapley-based search, yet incurs thousands of pipeline evaluations and offers limited transfer. Neither explicitly exploits dataset semantics for pruning and adaptation.

Differentiable Construction. Conversely, this paradigm relaxes the discrete search problem into continuous optimization. WindTunnel [59] and DiffML [21] model pipelines as differentiable tensor computations or weighted mixtures, allowing gradients to backpropagate from the downstream model. DiffPrep [31] further formalizes this by enabling the learning of transformation hyperparameters via differentiable relaxations. Although these methods theoretically allow for fine-grained optimization, they are constrained to differentiable operators and incur substantial computational overhead due to the requirement of repetitive end-to-end gradient updates, limiting their scalability on large heterogeneous datasets.

2.2 Reinforcement Learning

RL serves as a foundational paradigm for combinatorial optimization in AutoML [57]. Policy Gradient methods such as PPO [45] dominate domains characterized by low-latency simulators [42]. However, they are ill-suited for data preparation where the reward signal is both sparse and computationally expensive to obtain. Since these on-policy algorithms require fresh trajectories for every gradient update, their inherent low sample efficiency renders them computationally infeasible when each evaluation necessitates time-consuming downstream model training.

Consequently, value-based methods, particularly DQN [52] and its variants, have become the standard for modeling data preparation as a Markov Decision Process [25]. These approaches are favored for their ability to leverage off-policy mechanisms, which enable the efficient reuse of historical experiences [7]. Nevertheless, value maximization faces intrinsic limitations in this domain due to the delayed nature of supervision. As valid feedback is only received after executing the entire pipeline, Temporal Difference (TD) learning struggles to assign credit to early-stage decisions accurately [26]. This ambiguity often drives agents to converge prematurely to suboptimal local maxima.

To address these issues, Generative Flow Networks (GFlowNets) offers a paradigm shift from reward maximization to reward sampling [3, 4]. Unlike standard RL agents that seek a single optimal path, GFlowNets treat the generation process as a flow-matching problem to sample trajectories proportional to their reward [50]. This formulation theoretically ensures mode-covering exploration and effective credit assignment across long horizons without the prohibitive sampling costs of on-policy methods. While proven effective in scientific discovery tasks such as molecule generation [22, 23, 61], this flow-matching paradigm remains unexplored in automated data preparation. *Rather than directly applying it, FlowPipe tailors this architecture by instantiating a C-GFlowNets conditioned on distilled semantic information. This structural adaptation injects a semantic prior into the flow logic, ensuring diverse exploration that strictly adheres to rigid logical dependencies.*

Table 1. Summary of Notations.

Symbol	Description
$\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}$	Training and test datasets
$\mathcal{T}$	Set of component categories
$\mathcal{O}$	Set of concrete operators
Ω	Hierarchical search space of pipelines
P	Data preparation pipeline
π	Policy
$\mathcal{M}$	Downstream ML model
ϕ	Performance metric
R	Reward derived from test performance
τ	Generation trajectory

2.3 Foundation Models in Data Preparation

Recent efforts to integrate Foundation Models into data preparation aim to bridge the semantic gap left by purely statistical approaches.

Generative paradigms [38, 62] leverage Large Language Model (LLM) to synthesize cleaning scripts directly via dialogue or few-shot prompting [32]. For instance, ChatPipe [9] employs dialogue-driven interaction, while CatDB [14] utilizes refined metadata from data catalogs and prompt chaining to guide code generation. However, these models primarily function as probabilistic token generators guided by static rules rather than rigorous optimizers, resulting in operational hallucinations and execution failures.

Alternatively, **representation-enhanced methods** such as CtxPipe [16] utilize pre-trained embeddings (*i.e.*, GTE-large [34]) to augment the state space of RL agents. Nevertheless, these systems typically rely on lossy data snapshots due to context limits [58] and employ shallow additive fusion that fails to capture the deep multiplicative dependencies between dataset semantics and operator logic. *Our work distinguishes itself by integrating LLM-enhanced priors via a rigorous GFlowNets formulation with FiLM-based modulation to achieve robust, semantic-aware pipeline synthesis.*

3 Problem Definition

In this section, we formalize the automated data preparation problem and introduce our formulation of pipeline synthesis as a generative flow process. Table 1 summarizes the frequently used notations.

To tackle the combinatorial explosion of pipeline configurations, we structure the search space into a two-level hierarchy rather than a flat list. By decoupling abstract functional primitives (*e.g.*, *Imputation*) from concrete instantiations (*e.g.*, *Mean*), this design separates functional intent from implementation—analogueous to logical versus physical operators—enabling early pruning of vast suboptimal search regions.

We formally define this structure as follows:

DEFINITION 1 (HIERARCHICAL OPERATOR SPACE). *Let $\mathcal{D}$ be the dataset. We define the search space Ω as a hierarchical collection of operations comprising two levels:*

- *Component Categories $\mathcal{T} = \{T_1, \dots, T_k\}$: These represent abstract functional primitives, such as *Imputers* and *Encoders*.*
- *Concrete Operators $\mathcal{O}$: Each category T_i maps to a specific subset of executable algorithms $\mathcal{O}_{T_i}$. For example, $\mathcal{O}_{IM} = \{\text{Mean}, \text{Median}, \text{MostFreq}\}$.*

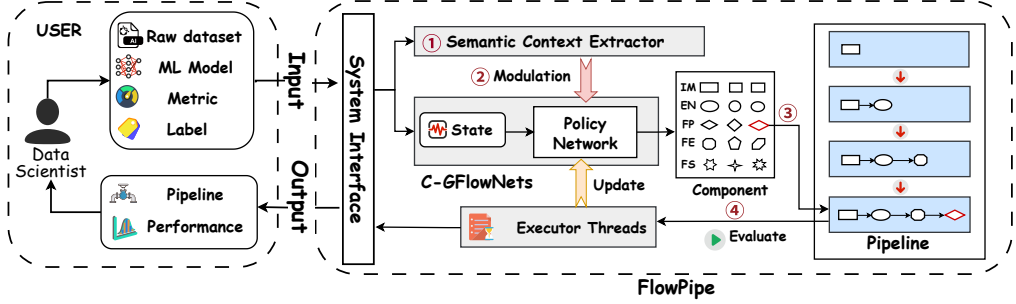


Fig. 2. **The workflow of FlowPipe.** It illustrates the high-level interaction between the user and system, ranging from the initial request to the final pipeline output.

Based on this hierarchy, a valid pipeline P is formally defined as a linear sequence of executable operators instantiated from $\mathcal{O}$. Concretely, P manifests as a sequential transformation chain: the input data $\mathcal{D}$ is passed through a specific operator selected from $\mathcal{O}_{\text{IM}}$ (e.g., MeanImputer) to handle missing values, and the transformed output is subsequently fed into an operator from $\mathcal{O}_{\text{EN}}$ (e.g., OneHotEncoder) for feature encoding, strictly adhering to the sequential input-output dependencies defined by the operator logic.

To leverage the generative capabilities of GFlowNets, we reformulate the static pipeline selection into a sequential decision-making process modeled as a flow network.

DEFINITION 2 (GENERATIVE TRAJECTORY). We model the construction of a data preparation pipeline P as a sequential trajectory $\tau = (s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots \xrightarrow{a_{f-1}} s_f)$. Here, s_0 denotes the initial state, representing an empty pipeline. At each discrete time step n , the agent selects an action a_n , defined as appending an operator from the candidate set $\mathcal{O}$ that is valid under current constraints, to transition to a new state s_{n+1} which encapsulates the updated pipeline topology. The process concludes at a terminal state s_f , corresponding to the fully constructed pipeline P . Each complete trajectory τ uniquely induces a pipeline $P(\tau)$.

As depicted in the C-GFlowNets of Figure 3, this trajectory generation is visualized as a directed path-finding process. The red dashed arrow (e.g., $s_3 \rightarrow s_4$) highlights the active transition step where the agent samples the next operator based on the current policy.

DEFINITION 3 (FLOW CONSISTENCY). The core objective is to learn a policy that satisfies Flow Consistency: The GFlowNets agent learns a stochastic policy π to generate trajectories such that the trajectory likelihood $p(\tau)$ of sampling a complete trajectory is directly proportional to its performance reward $R(P)$:

$$p(\tau; \theta) \propto R(P; \mathcal{D}). \quad (1)$$

This proportionality ensures that high-performing pipelines are sampled with higher probability. Unlike value-based RL formulations that primarily focus on optimizing a single trajectory, flow consistency distributes probability mass across multiple high-reward pipelines.

Finally, based on the definitions above, we formally state the optimization problem.

Let $\mathcal{P}(\Omega)$ denote the set of valid pipeline sequences constructible from the hierarchical operator space Ω .

PROBLEM (REWARD-MAXIMIZING PIPELINE GENERATION). Given a dataset $\mathcal{D}$ and a downstream model $\mathcal{M}$, we aim to learn a generative policy π_θ parameterized by θ .

While the training phase aligns the generation probability with the reward distribution (Eq. (1)), the practical inference objective is to identify the optimal pipeline P^* . This entails finding the configuration

that maximizes the validation reward R , approximated by sampling trajectories τ from the trained policy π_θ :

$$P^* = \underset{P \in \mathcal{P}(\Omega)}{\operatorname{argmax}} R(P; \mathcal{D}) \approx \underset{\tau \sim \pi_\theta}{\operatorname{argmax}} R(P(\tau); \mathcal{D}). \quad (2)$$

4 System Overview

Figure 2 depicts the architecture of **FlowPipe**. The system accepts a raw dataset $\mathcal{D}$ and a machine learning task $\mathcal{T}$ (comprising the target model, metric, and labels) as input, and synthesizes an executable data preparation pipeline P to maximize downstream performance.

Unlike prior RL-based systems that rely on greedy action-value policies, FlowPipe synergizes semantic reasoning with generative flow modeling. By formulating pipeline construction as a probabilistic inference task via a C-GFlowNets, and injecting global semantic priors through FiLM-based modulation, FlowPipe steers the exploration toward semantically plausible and high-performing regions of the operator space. The detailed workflow proceeds as follows.

Step 1 - Semantic Context Extraction. The workflow commences at the *System Interface*, which parses the input schema. To address the limitation where statistical profiles fail to capture operational logic, the *Semantic Context Extractor* constructs a Structured Prompt and distills it into a dense modulation vector $\mathbf{V}_{sc}$ via an LLM.

Example: Consider a used car dataset containing a “Zip_Code” column. While standard statistics merely see a high-variance numerical distribution, our Extractor captures its semantic identity as a nominal geospatial identifier. This domain-aware insight allows the system to preemptively prune magnitude-based operations (e.g., Scaling), thereby significantly reducing the combinatorial search space by filtering out semantically incoherent trajectories. Details on the prompt construction and extraction strategy are in Section 5.

Step 2 - Semantic-Modulated Policy Initialization. Before the search phase, the Modulation vector is injected into the *Policy Network* to guide the agent. We employ a FiLM mechanism to recalibrate the policy’s activations, effectively pruning operationally invalid actions based on the distilled semantics.

Example: For the “Zip_Code” feature, the modulation signal suppresses the logits for Feature Preprocessing operators (e.g., Min Max Scaler), as scaling a categorical code is semantically invalid. Conversely, it amplifies the probability of valid Encoder (EN) operators like Label Encoder. This narrows the search space and ensures logical correctness. The architectural implementation of this module is detailed in Section 6.3.

Step 3 - Generative Pipeline Synthesis. The C-GFlowNets engine instantiates a sequential decision process, visualized as the state transition diagram at the bottom of Figure 3. At each step, the *Semantic-Modulated Policy* observes the current state and samples a specific component proportional to its learned flow probability. Specifically, the red dashed lines highlight an active decision instance where the policy evaluates state s_3 and samples an action to transition to s_4 . We elaborate on the state representation and probabilistic modeling in Section 6.

Example: Guided by the policy, the agent samples a multi-hop transformation trajectory for “Zip_Code”. It first selects Target Encoder to convert the high-cardinality categories into numerical risk scores, and subsequently chains Robust Scaler to normalize these scores. This sequential generation demonstrates FlowPipe’s capability to build complex, topologically coherent workflows rather than isolated operator choices. We elaborate on the state representation and probabilistic modeling in Section 6.

Step 4 - Execution and Evaluation. Generated pipeline candidates are dispatched to *Executor Threads*, which materialize transformations in an isolated runtime. The system trains the downstream machine learning model to compute a validation metric, serving as the terminal reward

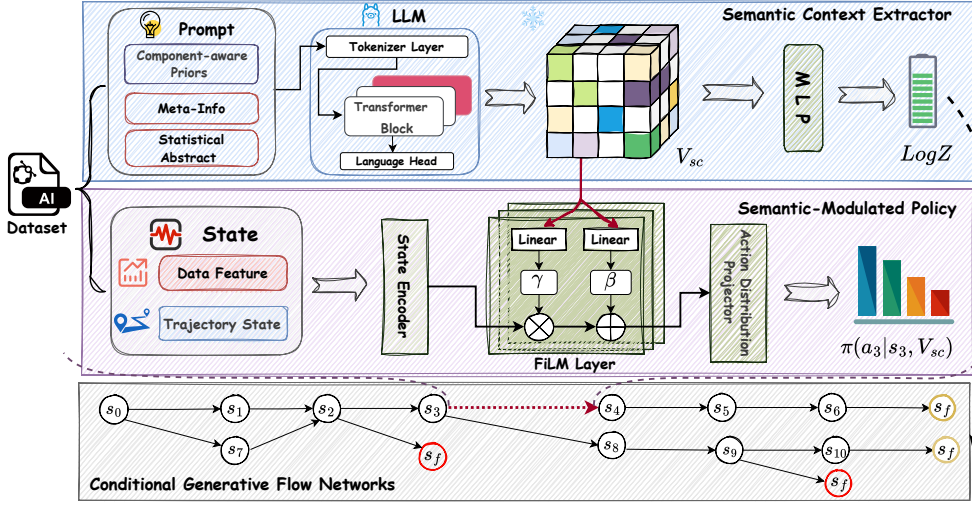


Fig. 3. **Overview of the Decision-Making Process.** A Generative LLM encodes the dataset into a semantic prior V_{sc} and normalization term $LogZ$. This prior modulates the policy network via FiLM layers (middle) to condition the sequential pipeline synthesis (bottom), exemplified by the specific action sampling from s_3 to s_4 (red dashed lines).

$R(\tau)$. This reward is used to update the policy via the Trajectory Balance (TB) [39] objective, which utilizes the normalization term $LogZ$ passed from the Semantic Context Extractor (as indicated by the dashed arrow in Figure 3). The detailed loss function is provided in Section 6.4.

In the subsequent sections, we detail the technical realization of these modules. Sec. 5 elaborates on the *Semantic Context Extractor* (Step 1), focusing on the LLM-driven derivation of semantic priors. Section 6 presents the C-GFlowNets framework. Crucially, this framework serves as the unified engine underpinning Steps 2 – 4: it integrates the semantic modulation mechanism (Step 2), drives the probabilistic synthesis process (Step 3), and governs the reward-based optimization via the Trajectory Balance objective (Step 4).

5 Semantic Context Extractor

As highlighted in Step 1 of the System Overview, standard statistical profiles often fail to capture the operational logic required for effective data preparation. Recent automated pipeline construction methods that incorporate embedding models, such as GTE [34], typically generate local static representations derived from limited data snapshots including random row samples. Consequently, these approaches suffer from a significant lack of semantic information and fail to capture the global semantic context required to deduce operational logic.

To effectively distill this global semantic context, we construct the Generative Semantic Context Vector through a two-step process. We first transform the raw dataset into a coherent textual description via a Structured Semantic Prompt, encapsulating its intrinsic semantics and operational constraints. Subsequently, by processing this prompt through a Generative LLM with a truncation-based extraction strategy, we distill the model’s latent reasoning into a fixed dense embedding. This embedding bridges the semantic gap by aligning intrinsic data profiles with operational logical priors. Acting as a global conditional prior, it provides the necessary inductive bias to dynamically modulate the RL state space and guide the GFlowNets training process, ensuring the optimization prioritizes decisions that are both data-compatible and operationally valid.

5.1 Structured Semantic Prompt

Constructing an effective semantic representation for large-scale tabular data presents a dual challenge. First, directly feeding raw datasets into LLMs is computationally prohibitive due to finite context window constraints. Second, heuristic approximations adopted by baselines like CtxPipe, which rely on randomly subsampling small fragments, suffer from severe *local sampling bias*. Consequently, these methods fail to capture critical global properties including class imbalance and long-tail distributions. Furthermore, raw data ingestion lacks the necessary task-specific inductive bias. Without explicit instructions, the LLM remains agnostic to the underlying optimization objective. To overcome these limitations, we explicitly construct a structured prompt $\mathcal{P}$ that functions as a mechanism for *lossless semantic compression*. To ensure scalability, this architecture explicitly excludes raw data rows, completely decoupling token complexity from dataset volume. Specifically, by relying on schema-level metadata and dataset statistics summaries, scaling to millions of instances introduces zero additional tokens, while the footprint of increasing feature dimensionality remains strictly marginal. Specifically, this prompt is composed of three functional blocks designed to concretize intrinsic semantics and operational constraints.

First, the **Meta-Info** block functions as the semantic anchor of the prompt. It aggregates explicit metadata, including the dataset name, attribute headers, and basic task properties, to construct a high-level identity profile of the data. These textual identifiers, such as column names like “credit_score”, encapsulate rich semantic cues that enable the LLM to infer domain-specific strategies that escape shallow statistical analysis. For instance, distinguishing “systolic_bp” from “credit_score” helps the model prioritize medical interpretability over financial noise tolerance. Critically, since this block targets only schema-level headers, its token count remains invariant to row volume. Increasing dimensionality (D) introduces only marginal overhead, as each additional feature merely appends a single textual identifier (e.g., the column name) to the sequence.

Complementing this anchor, the **Statistical Abstract** synthesizes a holistic distributional profile. It translates discrete numerical properties into a serialized schema narrative. Designed as a compact set of critical descriptors, this block captures the topological features defining dataset quality. Specifically, it enumerates per-column statistics including central tendency metrics like mean and mode, dispersion indicators such as range, and missingness ratios to expose structural constraints. These signals allow the model to assess dataset density for imputation and arbitrate between encoding strategies based on cardinality. Moreover, it explicitly maps strong pairwise correlations where the absolute Pearson coefficient $|\rho|$ exceeds 0.5, revealing intrinsic feature space redundancy to guide feature selection. Likewise, our design only uses the statistics collected from the dataset rather than involving raw data instances. Therefore, this block’s length is completely unaffected by row volume, requiring only a concise statistical profile for each added feature.

Crucially, we inject **Component-aware Priors** into the prompt to bridge the semantic gap between static statistical profiles and dynamic operator selection. This block functions as a logical instruction set that explicitly maps serialized data characteristics to operator preconditions. By linguistically articulating applicability heuristics, such as specifying that significant feature redundancy warrants PCA and high-cardinality categories necessitate Target Encoding, we establish a basis for the subsequent LLM encoding phase. This formulation strictly constructs the logical context to guide the downstream reasoning, ensuring that the latent representation generated later is constrained by operational validity. Importantly, the information injected into this block is entirely independent of the dataset’s volume and dimensionality. Therefore, its token footprint remains constant, ensuring scalability with zero additional overhead as the dataset grows.

5.2 LLM-Enhanced Latent Reasoning

Given the structured prompt $\mathcal{P}$, our objective is to distill the LLM’s latent reasoning regarding the alignment of intrinsic data properties with operational prerequisites into a dense, operable vector.

Standard discriminative embedding models like BERT [12] or GTE [34] are constrained by optimization objectives that prioritize static lexical similarity. While effective for matching semantically related terms, they lack the causal reasoning mechanism required to process conditional logic. For instance, GTE may relate “skewness” to “distribution” but fails to infer that high skewness necessitates Log-Transformation. To address this limitation, we leverage the context-adaptive reasoning capacity of the Generative LLM to preserve the continuous reasoning state implicit in the model activations rather than collapsing it into discrete tokens.

Structurally, the Generative LLM follows a standard decoder-only transformer architecture comprising a stack of K blocks followed by a Language Model head. We forward-propagate the prompt $\mathcal{P}$ through the entire transformer stack but truncate the execution explicitly at the output of the K -th block to bypass the LM head. Instead of projecting the hidden states into the high-dimensional vocabulary space for token generation, we extract the raw output features from the final transformer layer, denoted as $\mathbf{H} \in \mathbb{R}^{L \times d}$ for sequence length L and hidden dimension d . We deliberately employ open-weights models to ensure white-box access to these internal representations. This design choice is critical for system deployment: it allows FlowPipe to bypass the high-latency iterative decoding process typical of black-box APIs, while ensuring data privacy by performing all semantic reasoning within the local runtime environment.

To resolve the variable sequence length driven by the feature dimensionality D , we apply mean pooling over the token dimension of $\mathbf{H}$. This operation compresses the high-dimensional semantic interactions into a compact, fixed-size manifold $\mathbf{V}_{sc}$. Crucially, producing a constant-sized representation decouples the LLM’s verbosity from the downstream policy, natively preventing the curse of dimensionality on wide tabular datasets. Computationally, this design strictly separates heavy semantic reasoning from efficient structure search. Furthermore, $\mathbf{V}_{sc}$ is generated and cached offline as a static global prior, amortizing the LLM’s inference cost and ensuring the iterative online search remains lightweight without recurrent overhead.

6 Conditional Generative Flow Networks

Having distilled the intrinsic data semantics into the modulation vector $\mathbf{V}_{sc}$, we now introduce the generative engine that leverages this prior. The core generative engine of FlowPipe is instantiated as a C-GFlowNets. Unlike standard RL agents that focus on optimizing a single high-reward trajectory, C-GFlowNets samples pipelines proportional to their reward to ensure diverse exploration of high-performance configurations.

6.1 Probabilistic Modeling Instantiation

Building upon the generative trajectory formulation in Definition 2, we instantiate the pipeline construction process as a linear sequential decision-making task, explicitly conditioned on the global semantic context $\mathbf{V}_{sc}$. In FlowPipe, the C-GFlowNets is conditioned on a global semantic vector $\mathbf{V}_{sc}$, which captures dataset-specific constraints and guides the generation of valid pipeline trajectories.

Formally, we parametrize a policy $\pi(s_{n+1}|s_n, \mathbf{V}_{sc}; \theta)$ with learnable parameters θ . The core objective is to learn a policy that satisfies the Flow Consistency condition, where the marginal likelihood $p(\tau|\mathbf{V}_{sc})$ of generating a complete trajectory τ is directly proportional to its terminal reward $R(\tau)$:

$$p(\tau | \mathbf{V}_{sc}) \propto R(\tau). \quad (3)$$

This unified probabilistic formulation mitigates the structural dissonance inherent in fragmented Multi-DQN architectures, thereby improving the credit assignment problem. By treating the generation process as a coherent flow, it compels the agent to internalize the global dependency between early-stage decisions (e.g., imputation) and downstream outcomes (e.g., classification accuracy) within a single policy π . Furthermore, conditioning the entire flow on $\mathbf{V}_{sc}$ transforms the optimization from a blind combinatorial search into a semantic-aware probabilistic sampling process, effectively constraining exploration to the high-probability manifold defined by the data's intrinsic properties.

6.2 State Representation

A robust state representation is critical for accurate environmental perception. At each decision step n , the agent observes a composite state s_n that encapsulates both the sequential history of the pipeline and the evolving statistical properties of the data. As shown in the State Block of Figure 3, we construct s_n via the concatenation of two distinct feature vectors $s_n = [\mathbf{s}_n^{tr} \oplus \mathbf{s}_n^d]$. Crucially, because each applied operator immediately mutates the underlying dataset and extends the pipeline history, both $\mathbf{s}_n^{tr}$ and $\mathbf{s}_n^d$ are dynamically recomputed at every transition step to ensure the policy evaluates strictly up-to-date environmental feedback.

Trajectory State ($\mathbf{s}_n^{tr}$). This vector encodes the **sequential context** of the partial pipeline. To avoid the computational latency associated with recurrent sequence modelers, such as LSTMs, we implement an efficient *Explicit Sequential Encoding*. We map the current pipeline configuration to a fixed-length binary vector $\mathbf{s}_n^{tr} \in \{0, 1\}^{|T|}$, where each dimension functions as an indicator for a specific operator category. For instance, given a vocabulary ordered as {Imputer, Encoder, ...}, a pipeline that has already applied an imputer is represented as [1, 0, ...]. This lightweight representation provides an instantaneous structural snapshot, enabling the policy to strictly enforce sequential constraints, such as prohibiting redundant imputation, via low-cost logical masking.

Data Feature State ($\mathbf{s}_n^d$). This vector captures the granular statistical profile of the intermediate dataset transformed by the previous n operators. It includes meta-features such as skewness, kurtosis, sparsity, and correlation coefficients. A key challenge is the vast numerical range of these statistics where cardinality can range from 2 to 10^6 while variance remains infinitesimal. Feeding raw values directly into neural networks often leads to gradient instability and poor convergence. To mitigate this, we utilize a *Dual-Channel Numerical Embedding*. We decompose each continuous meta-feature ψ into a tuple representing its magnitude and precision via the mapping $\psi \rightarrow (\text{frac}(\psi), \exp(\psi))$. This representation effectively disentangles numerical scale from value distribution to prevent information collapse for outlier values and ensure stable training across heterogeneous datasets.

6.3 Semantic-Modulated Policy Network

The Semantic-Modulated Policy serves as the decision-making kernel. It maps the composite state s_n to a probability distribution over the operator space $\mathcal{O}$.

Context-aware baselines such as CtxPipe rely on *shallow additive fusion*. As highlighted by the problem of semantic detachment, this formulation functions merely as an *extrinsic bias term*, which is structurally insufficient to capture the deep, multiplicative dependencies between data profiles and operator logic. To resolve this, we employ FiLM for deep semantic modulation. To resolve this, we adopt Feature-wise Linear Modulation (FiLM) [41]. While recognized as an established neural conditioning technique, we exploit its structural properties specifically for deep semantic modulation in data preparation. By utilizing $\mathbf{V}_{sc}$ to multiplicatively recalibrate feature activations, this mechanism functions as a differentiable logic gate. It enables dynamic logic adaptation and performs the soft action suppression essential for mitigating *exploration inefficiency*.

FiLM-Based Semantic Modulation. We implement semantic injection via a FiLM mechanism. We first project the raw state s_t into a latent feature sequence h_t . Simultaneously, the invariant context $\mathbf{V}_{sc}$ is mapped to layer-specific affine parameters via a projection head $g^{(l)}$. For the l -th layer of the policy network, the feature modulation is defined as:

$$h_n^l = \text{FiLM}(h_n^{l-1} \mid \mathbf{V}_{sc}) = \gamma^l \odot h_n^{l-1} + \beta^l, \quad (4)$$

here $\odot$ denotes the element-wise Hadamard product. The projection head $g^{(l)}$ consists of two separate multilayer perceptrons that predict the scaling vector γ^l and shifting vector β^l respectively.

This design enables a dual-mechanism control over the decision logic. The multiplicative coefficient $\gamma^l(\mathbf{V}_{sc})$ functions as a dynamic feature re-calibration mechanism. It selectively amplifies or suppresses specific feature channels based on the logical priors encoded in $\mathbf{V}_{sc}$. Conversely, the additive coefficient β^l introduces a context-dependent bias shift, adjusting the activation thresholds of operators to align with data properties. By applying this modulation recursively across all L layers of the policy network, FlowPipe achieves hierarchical semantic alignment. Early layers effectively prune low-level incompatible signals while deeper layers refine abstract operator sequencing strategies.

Crucially, this mechanism implements the *Soft Action Suppression* required to mitigate exploration inefficiency. Revisiting the “Zip_Code” case where $\mathbf{V}_{sc}$ encodes a nominal constraint, the projection head learns to predict near-zero γ coefficients for magnitude-dependent operators, such as MinMaxScaler. This operation effectively suppresses forward signal propagation for these semantically incompatible actions and significantly reduces their gradient contributions. Consequently, the agent is guided to explore valid regions of the operator space, which reduces exploration variance and accelerates convergence compared to unconstrained exploration in standard RL.

Action Distribution Modeling. The semantically modulated feature vector is passed to an *Action Distribution Projector* which maps the latent representation to logits $\tilde{L}_n$ over the operator space $\mathcal{O}$. In contrast to value-based baselines (e.g., DQN) that often induce near-deterministic behavior via greedy action selection and to support robust pipeline discovery under a large combinatorial search space, we model the action choice as a Categorical Distribution:

$$a_n \sim \pi(a_n = k \mid s_n, \mathbf{V}_{sc}) = \text{Softmax}(\tilde{L}_n) = \frac{\exp(\tilde{L}_n^k)}{\sum_{j \in \mathcal{O}} \exp(\tilde{L}_n^j)}, \quad (5)$$

where k represents the operator chosen by the policy at step n , while j enumerates all candidate operators in $\mathcal{O}$. This stochastic sampling mechanism prevents premature mode collapse and supports the discovery of multiple high-quality pipeline configurations.

6.4 Training Objective

Training faces delayed, expensive end-to-end evaluation feedback. We thus adopt the TB objective for Conditional GFlowNets, where the semantic context $\mathbf{V}_{sc}$ conditions the trajectory distribution via a partition function.

Trajectory-Level Supervision for Credit Attribution. Value-based RL methods typically rely on step-wise bootstrapping, which often propagates delayed validation signals inefficiently over long pipeline horizons, making it difficult to attribute final performance to early-stage decisions. In contrast, TB-style flow matching enforces trajectory-level consistency and provides holistic supervision that relates the terminal reward $R(\tau)$ to all decisions along the trajectory. As a result, the objective induces an implicit gradient path from the terminal validation reward back to early pipeline decisions, without relying on step-wise bootstrapping.

Formally, following the Conditional GFlowNet formulation, we obtain the context-dependent partition function $\log \mathbf{Z}$ by mapping the semantic condition vector $\mathbf{V}_{sc}$ through a lightweight MLP.

Table 2. Selected components and their types.

Component Category	Operators
Imputer (IM)	Mean, Median, Most Frequent
Encoder (EN)	Numeric Data, Label Encoder, One-hot Encoder
Feature Preprocessing (FP)	Min Max Scaler, Max Absolute Scaler, Robust Scaler, Standard Scaler, Quantile Transformer, Normalizer, Power Transformer, K-bins Discretizer
Feature Engineering (FE)	Polynomial Features, Interaction Features, PCA, Kernel PCA, Incremental PCA, Truncated SVD, Random Trees Embedding
Feature Selection (FS)	Variance Threshold

In addition, because the combinatorial search space contains many invalid or incomplete pipelines that yield no reward, we introduce a *failure-aware weighting* scheme to limit their influence during optimization. Specifically, we minimize the following TB-style loss:

$$\mathcal{L}(\tau; \theta) = w(\tau) \cdot \left(\log Z + \sum_{n=0}^{|\tau|-1} \log \pi(a_n \mid s_n; \mathbf{V}_{sc}) - \log R(\tau) \right)^2. \quad (6)$$

This objective combines three complementary components. First, $\log Z$ provides context-conditioned normalization, which stabilizes learning across datasets with heterogeneous reward scales. Second, the trajectory-level log-probability term distributes supervision across all pipeline decisions, improving credit attribution with respect to the final validation reward. Finally, $w(\tau)$ implements failure-aware weighting: it assigns unit weight to valid pipelines and a reduced penalty factor $\lambda \in [0, 1)$ to invalid or incomplete executions, thereby focusing learning on valid, high-quality pipeline configurations. All parameters θ are jointly optimized by minimizing this flow-matching loss.

7 Experiment

Our experiments target answering the following four questions:

- **RQ1 (Effectiveness):** How does FlowPipe compare with SOTA baselines across real-world datasets?
- **RQ2 (Efficiency):** How do FlowPipe's offline training cost and online latency compare with search-based and meta-learning methods?
- **RQ3 (Component Analysis):** How do LLM-based semantic reasoning and FiLM modulation contribute to performance?
- **RQ4 (Scalability & Robustness):** How does FlowPipe scale with dataset size and respond to hyperparameter variations?

7.1 Experimental Setup

Hardware and OS. Experiments are conducted on an Ubuntu 22.04 server with dual AMD EPYC 7402 CPUs (48 cores), 256GB RAM, and NVIDIA RTX 4090 GPUs (CUDA 12.2). For SAGA, the Java heap size is set to 80GB.

Implementation Details. For the semantic context extraction module, we employ **Llama-3.1-8B-Instruct** [17] as the backbone Large Language Model. The model is frozen during the GFlowNet training phase, serving solely as a static feature extractor to distill reasoning-rich semantic vectors. All neural network components within the GFlowNet agent are implemented using PyTorch.

Datasets. We use a comprehensive suite of real-world datasets sourced from OpenML [53], UCI [2], and Kaggle. To comprehensively evaluate scalability and out-of-distribution generalization,

Table 3. Performance on the DiffPrep dataset collection. The **Improvement** row denotes FlowPipe’s gain over SOTA.

Setup	DiffPrep Dataset Collection					
	Test Accuracy ↑	Ranking ↓	Precision ↑	Recall ↑	F1-score ↑	Inference Time (s) ↓
DEF (<i>SIGMOD’23</i>)	0.724	9.512	0.605	0.629	0.604	36.237
RS (<i>SIGMOD’23</i>)	0.761	8.125	0.647	0.673	0.650	803.888
DP-Fix (<i>SIGMOD’23</i>)	0.780	6.422	0.681	0.727	0.687	1024.845
DP-Flex (<i>SIGMOD’23</i>)	0.784	4.188	0.697	0.722	0.695	11148.911
DL (<i>SIGKDD’20</i>)	0.704	11.115	0.616	0.584	0.570	<u>18.923</u>
HAI-AI (<i>SIGMOD’23</i>)	0.760	8.194	0.705	0.619	0.613	22.195
SAGA (<i>SIGMOD’23</i>)	0.731	8.286	0.647	0.640	0.618	1384.824
TPE (<i>arXiv’23</i>)	0.781	4.580	0.633	0.644	0.637	1299.110
Direct Prompting	0.712	10.420	0.623	0.631	0.615	142.508
ChatPipe (<i>SIGMOD’24</i>)	0.783	4.310	0.751	0.718	0.732	278.441
CtxPipe (<i>SIGMOD’24</i>)	<u>0.806</u>	<u>3.655</u>	<u>0.774</u>	<u>0.736</u>	<u>0.745</u>	65.203
SwiftDP (<i>ICDE’25</i>)	0.728	9.182	0.700	0.702	0.683	10.203
FlowPipe (Ours)	0.896	1.139	0.788	0.794	0.780	51.516
Improvement	+11.17%	+2.516	+1.81%	+7.88%	+4.70%	–

these datasets encompass a diverse range of feature dimensionalities ($D \in [4, 343]$) and are strictly separated into disjoint sets. We train our agent on the HAIPipe dataset collection [10] (353 datasets) and perform zero-shot evaluation on the DiffPrep [31] (18 datasets) and DeepLine [20] (56 datasets) collections. We ensure zero overlap between the training and test collections to verify the model’s capability to synthesize pipelines for entirely unseen tasks.

Components. All components adopt the scikit-learn API (`fit()` and `transform()`) for broad compatibility and seamless integration of User-Defined Functions (UDFs). Our default search space (Table 2) aligns with SOTA benchmarks [10, 16]. Furthermore, the architecture natively supports broader data preparation operations [19, 37], facilitating the continuous expansion of *macro-categories* and *micro-operators* without combinatorial explosion. Topologically, we employ a flexible linear architecture that allows the agent to dynamically select, reorder, or bypass components to discover non-trivial sequences.

Downstream Model. Adhering to standard evaluation protocols in automated data preparation [16], we utilize Logistic Regression as the downstream evaluator. This choice ensures rigorous baseline comparison and guarantees broad reproducibility, as the model is natively supported by major frameworks including PyTorch, TensorFlow, scikit-learn, and Apache SystemDS [6].

Evaluation Metrics. We adopt Test Accuracy as the primary metric ϕ to align with baseline protocols, supplemented by Macro-Precision, Recall, and F1-score. We select macro-averaging because micro-averaged metrics are mathematically equivalent to accuracy in our setup. To assess efficiency, we report Pipeline Runtime, defined as the end-to-end wall-clock time comprising pipeline construction, execution, and downstream model training. This metric is critical for the evolving data analytics lifecycle, where distribution shifts [28, 36] necessitate frequent pipeline reconstruction.

7.2 Performance Comparison

Tables 3 and 4 present the **average performance** comparison results against baselines on the DiffPrep and DeepLine dataset collection, respectively. Detailed performance breakdowns for individual datasets are provided in the Appendix.

Table 4. Performance on the DeepLine dataset collection. The **Improvement** row denotes FlowPipe’s gain over SOTA.

Setup	DeepLine Dataset Collection					
	Test Accuracy ↑	Ranking ↓	Precision ↑	Recall ↑	F1-score ↑	Inference Time (s) ↓
DEF (<i>SIGMOD’23</i>)	0.760	8.722	0.754	0.769	0.751	268.899
RS (<i>SIGMOD’23</i>)	0.776	8.328	0.755	0.776	0.751	2069.028
DP-Fix (<i>SIGMOD’23</i>)	0.780	7.356	0.767	0.780	0.764	325.648
DP-Flex (<i>SIGMOD’23</i>)	0.789	6.122	0.772	0.789	0.769	2183.917
DL (<i>SIGKDD’20</i>)	0.740	9.921	0.738	0.740	0.727	48.439
HAI-AI (<i>SIGMOD’23</i>)	0.801	5.189	0.781	0.753	0.767	16.780
SAGA (<i>SIGMOD’23</i>)	0.809	<u>4.132</u>	0.714	0.731	0.723	784.132
TPE (<i>arXiv’23</i>)	0.792	5.850	0.704	0.711	0.708	382.341
Direct Prompting	0.723	11.450	0.619	0.611	0.615	112.107
ChatPipe (<i>SIGMOD’24</i>)	0.777	7.620	0.741	0.718	0.730	284.411
CtxPipe (<i>SIGMOD’24</i>)	<u>0.813</u>	5.611	<u>0.784</u>	<u>0.787</u>	<u>0.775</u>	14.310
SwiftDP (<i>ICDE’25</i>)	0.780	6.845	0.692	0.636	0.643	5.760
FlowPipe (Ours)	0.912	1.089	0.813	0.823	0.805	<u>10.510</u>
Improvement	+12.17%	+3.043	+3.70%	+4.57%	+3.87%	–

7.2.1 Baselines. We compare FlowPipe against the following SOTA methods. To ensure a fair comparison, we adopt their open-source implementations and strictly adhere to the hyperparameter configurations recommended in their original papers or documentation:

- **Default (DEF) [31]:** A standard rule-based pipeline aligned with commercial AutoML frameworks like Azure [24] and H2O [27]. It performs mean and mode imputation for numerical and categorical features, respectively, followed by standardization.
- **Random Search (RS) [31]:** Randomly instantiates 20 pipelines under the DP-Fix template, reporting the best-validated one.
- **DiffPrep-Fix (DP-Fix) and DiffPrep-Flex (DP-Flex) [31]:** We use the configurations reported in [31]. DP-Fix searches under a fixed logical pipeline structure, while DP-Flex performs gradient-based optimization over a continuous relaxation to enable flexible component selection.
- **DeepLine (DL) [20]:** A monolithic DQN baseline that uses a single Q-network for end-to-end decision making [20]. We use the authors’ implementation with the configuration in [20] and train for 150,000 steps, following authors’ recommendation.
- **HAIpipe (HAI-AI) [10]:** The AI-pipe agent from HAIpipe [10], built on a Multi-DQN architecture with stage-wise Q-networks for sequential pipeline generation. We use the authors’ released model trained for 56,000 steps.
- **SAGA [49]:** An evolutionary baseline using genetic programming for data cleaning pipeline optimization, configured as in [49].
- **TPE [54]:** A SOTA Bayesian optimization algorithm using density estimation and likelihood ratio for efficient black-box search.
- **Direct Prompting:** A zero-shot baseline where Llama-3.1-8B generates pipelines directly from FlowPipe’s semantic context.
- **ChatPipe [9]:** A conversational LLM baseline that generates pipelines for automated data preparation.
- **CtxPipe [16]:** A context-aware RL baseline that leverages pretrained language models to encode dataset semantics, and adopts a *Multi-DQN* architecture [16].

Table 5. Performance across different downstream models.

Benchmark	Downstream Model	CtxPipe	FlowPipe
DiffPrep	Random Forest (RF)	0.814	0.921
	MLP	0.778	0.887
DeepLine	Random Forest (RF)	0.829	0.871
	MLP	0.826	0.933

- **SwiftDP [30]**: An MCTS-based framework that combines meta-learning and attention to accelerate pipeline generation [30].

7.2.2 Analysis of Effectiveness (RQ1). We first evaluate the performance of constructed pipelines on test datasets for all setups. This is measured by the evaluation metrics ϕ and its ranking across setups per dataset.

Test Accuracy. Tables 3 and 4 summarize aggregated performance, with dataset-level breakdowns deferred to the **Appendix**. Evaluated rigorously across 10 random seeds, FlowPipe exhibits exceptional systemic stability with a negligible accuracy variance of strictly ± 0.003 , robustly outperforming deterministic baselines.

In aggregate, FlowPipe leads significantly with average accuracies of **0.896** on the DiffPrep collection and **0.912** on the DeepLine collection, surpassing the previous SOTA results of 0.806 and 0.813, respectively. These improvements are substantial in the context of data preparation [20, 31]. FlowPipe’s integration of contextual information results in significant performance gains, particularly on complex tasks. For instance, the *Accident* dataset shows a 41.3% relative improvement in accuracy compared to the best baseline (0.978 vs 0.692). Datasets such as *abalone*, *connect-4*, and *micro* also see notable improvements, benefiting from high-dimensional feature interactions or semantic dependencies, which FlowPipe’s LLM-driven context plug-in effectively captures to optimize performance.

Optimality Gap Analysis. We compare FlowPipe with an approximated exhaustive search (ES^*) utilizing 10,000 trials, a computationally intensive baseline consuming **30 hours** per dataset. In stark contrast, FlowPipe converges within **seconds** yet consistently matches or exceeds this strong baseline. Empirically, FlowPipe achieves optimal or superior configurations in **62 out of 74** evaluated cases (spanning **14/18** on DiffPrep and **48/56** on DeepLine). This confirms that even in high-dimensional spaces where brute-force enumeration fails, our semantic guidance effectively identifies global optima with orders-of-magnitude efficiency gains.

Setup Ranking. To analyze dataset-level effectiveness, we ranked the test accuracy of all 13 setups (Tables 3 and 4). FlowPipe significantly outperforms the second-best methods, achieving average ranks of 1.139 on the DiffPrep collection (vs. CtxPipe’s 3.655) and 1.089 on DeepLine (vs. SAGA’s 4.132). Furthermore, despite sharing similar component candidates with baselines, FlowPipe secures the #1 rank on approximately 78% of DiffPrep and over 85% of DeepLine datasets.

Other Metrics. Beyond simple accuracy, we evaluate macro-averaged precision, recall, and F1-scores to assess feature quality under class imbalance. FlowPipe achieves a superior macro F1-score of **0.780** on DiffPrep and **0.805** on DeepLine, significantly outperforming the strongest baseline, CtxPipe (0.745 and 0.775, respectively). These concurrent gains confirm that FlowPipe avoids majority-class overfitting; rather, it constructs highly discriminative pipelines that effectively resolve complex decision boundaries and minimize false positives better than prior methods.

Robustness Across Downstream Models. To prevent overfitting to the Logistic Regression evaluator, FlowPipe inherently optimizes across an ensemble of diverse predictors (e.g., Random Forest, MLP, Ridge, SGD). Table 5 verifies this robust generalization to non-linear architectures.

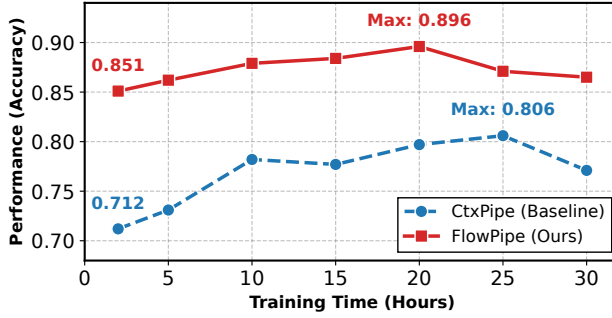


Fig. 4. Training efficiency comparison.

On DiffPrep, FlowPipe achieves average accuracies of 0.921 (Random Forest) and 0.887 (MLP), substantially outperforming CtxPipe (0.814 and 0.778, respectively). Similar dominance on DeepLine confirms that FlowPipe captures intrinsic data semantics to construct universally robust pipelines for a diverse spectrum of downstream models.

7.2.3 Case Study. To elucidate FlowPipe’s substantial performance margin over CtxPipe (0.819 vs. 0.590), we qualitatively dissect pipelines generated for the *google* dataset. This dataset represents a complex feature space characterized by severe power-law distributions (e.g., “Reviews”) and high-cardinality categoricals. The constructed pipelines reveal fundamental reasoning differences:

FlowPipe (Ours): Mean → QuantileTransformer → InteractionFeatures

CtxPipe (SOTA): Mean → LabelEncoder → VarianceThreshold → PowerTransformer → Random-TreesEmbedding

While both identify rudimentary data hygiene needs (e.g., Mean), subsequent selections diverge sharply. CtxPipe mechanically applies high-variance embeddings (e.g., Random-Trees-Embedding) without resolving underlying skewness. Conversely, FlowPipe’s FiLM mechanism leverages LLM priors to recognize power-law semantics in features like “Reviews”. It proactively prioritizes a Quantile-Transformer, effectively neutralizing long-tail effects via targeted semantic alignment. Beyond isolated operators, this divergence exposes flaws in sequential reasoning. CtxPipe’s decoupled Multi-DQN struggles with long-horizon credit assignment, inevitably trapping the search in sub-optimal deterministic paths. FlowPipe’s flow-based trajectory optimization overcomes this holistically, enforcing a critical prerequisite: stabilizing the feature distribution *before* applying Interaction-Features. This safely captures non-additive relationships without amplifying noise. Ultimately, FlowPipe transcends pure combinatorial search, performing *semantic navigation* tightly aligned with intrinsic data semantics.

7.3 Efficiency Analysis (RQ2)

Efficiency is paramount in production environments where continuous data streams require expedient processing [55]. FlowPipe addresses this by employing **amortized optimization**. Unlike evolutionary methods (e.g., SAGA) that incur high search costs for every new dataset, FlowPipe treats training as a one-time offline investment, enabling rapid online inference.

Training Efficiency and Amortization. Figure 4 illustrates the training efficiency on the DiffPrep collection. FlowPipe exhibits an extremely steep learning curve: it surpasses the peak performance of the strongest baseline, CtxPipe (which requires 25 hours), in just **2 hours**—a **12.5×** speedup. As training saturates at approximately 20 hours, FlowPipe extends its lead to a peak accuracy of **0.896**.

Online Inference Latency. We define *inference time* as the wall-clock latency to synthesize a pipeline for an unseen dataset. Computationally, FlowPipe’s generative state transitions scale

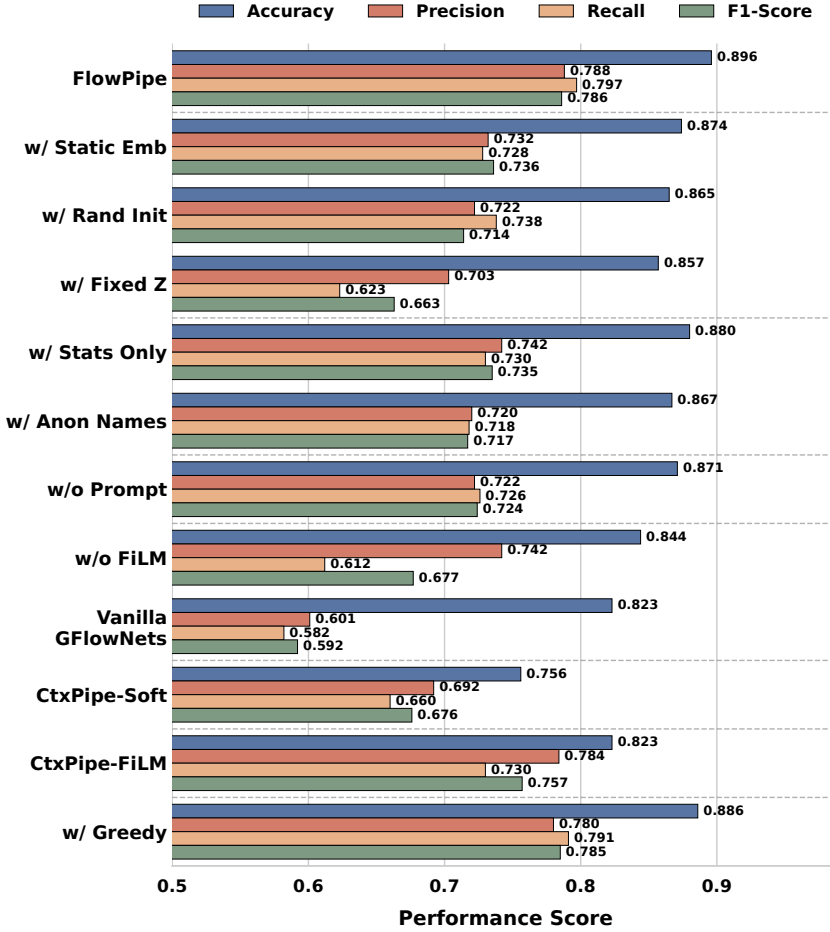


Fig. 5. Component ablation and baseline exploration.

linearly $O(L)$ with pipeline depth L . Crucially, our *decoupled prompt design* extracts strictly column metadata, rendering the generation complexity independent of dataset volume ($O(1)$ w.r.t. row count N). Scaling to millions of rows introduces zero additional tokens. Across dimensions up to *home_credit* ($D = 343$), our largest prompt consumes only $\sim 25\%$ of a 128K context window, ensuring robust scalability. To ensure rigorous evaluation, reported inference times *strictly include* the execution overhead of the Llama-3.1-8B model. Empirical profiling reveals this one-time offline extraction averages just 0.383s (1.86% of total latency). Furthermore, computing and caching these LLM priors offline grants FlowPipe a lightweight memory footprint of ~ 1.9 GB VRAM, vastly outperforming Multi-DQN baselines (e.g., CtxPipe at ~ 4.5 GB) that require resident models during the active RL loop.

On the DiffPrep collection, FlowPipe achieves an average latency of 51.516s, representing an order-of-magnitude acceleration over “search-at-inference” baselines like RS (803.9s) and SAGA (1384.8s). Similarly, on DeepLine, FlowPipe (10.51s) drastically outperforms the gradient-based DP-Flex (>2100 s). While SwiftDP achieves the lowest absolute latency (5.76s) via static retrieval,

this comes at a severe cost to quality (0.780 accuracy vs. 0.912 for FlowPipe), as its generalization is strictly bounded by repository coverage.

To conclude, FlowPipe establishes a win-win dominance over the strongest SOTA baseline CtxPipe. It simultaneously outperforms CtxPipe in effectiveness (surpassing average accuracy on both benchmarks) and efficiency. Specifically, FlowPipe reduces the inference latency from 65.203s (CtxPipe) to 51.516s on DiffPrep and from 14.310s to 10.51s on DeepLine, achieving substantial speedups of **20.99%** and **26.55%**, respectively. This confirms FlowPipe as the new Pareto-optimal solution for automated data preparation.

7.4 Ablation Study (RQ3)

To rigorously validate the contribution of each component in FlowPipe, we conduct a comprehensive ablation study on the DiffPrep collection. We isolate specific design choices through component replacements (w/) and removals (w/o), with comparative results visualized in Figure 5.

Effectiveness of Semantic Reasoning. Replacing the LLM reasoning module with static *GTE-Large* embeddings (w/ Static Emb [16]) degrades accuracy to 0.874. Similarly, using raw snapshots (w/o Prompt, 0.871) or random contexts (w/ Rand Init, 0.865) yields comparable drops. This shows passive representations fail to deduce actionable rules. Thus, these ablations confirm active LLM-derived priors are strictly essential to guarantee zero-shot generalization and prevent overfitting.

Beyond Statistical Compression. To verify that the LLM performs genuine reasoning rather than mere data compression, we test two prompt variants. Restricting the prompt to pure statistics (w/ Stats Only) degrades accuracy to 0.880. Crucially, retaining the full prompt structure but obfuscating explicit column names (w/ Anon Names) drops accuracy further to 0.867. This proves that statistical profiles alone cannot resolve operational ambiguities; the LLM actively relies on real-world semantic anchors to map statistics into valid transformation logic.

Impact of Deep Modulation Mechanism. We evaluate the structural integration of semantic priors. Reverting the FiLM mechanism to the shallow additive fusion used in prior works (w/o FiLM) results in the sharpest accuracy drop among semantic variants (0.844). This empirically validates that additive fusion is structurally inadequate for modeling complex conditional dependencies. FiLM functions as a differentiable logical gate, explicitly magnifying or suppressing feature channels based on context; its removal causes a severe loss of policy expressiveness.

Necessity of Dynamic Exploration Calibration. Replacing the learnable partition function $\log Z$ with a fixed scalar (w/ Fixed Z) degrades accuracy to 0.857 and, more critically, collapses the F1-Score to 0.663. This discrepancy validates our theoretical assertion regarding *Global Flow Calibration*. Without a dynamically calibrated exploration budget, the agent converges to a conservative equilibrium, prioritizing simple, safe pipelines while failing to explore the complex, high-reward architectures necessary for optimizing F1.

Structural Supremacy of Unified Flow Matching. The semantic-free Vanilla GFlowNets (**0.823**) outperforms prior RL baselines, including monolithic DQN (DeepLine: **0.704**) and Multi-DQN (CtxPipe: **0.806**). This isolates the structural advantage of our framework: unlike fragmented value-based methods that struggle with delayed rewards, trajectory-level flow-matching natively propagates terminal supervision back to early decisions, guaranteeing stable and efficient long-horizon credit assignment.

Insufficiency of Value-Based Augmentations. To verify our gains stem from a fundamental paradigm shift, we evaluate two targeted augmentations to the Multi-DQN baseline (Figure 5). First, appending FiLM for semantic integration (CtxPipe-FiLM) improves vanilla CtxPipe (0.823 vs. 0.806) but significantly trails FlowPipe (0.896), proving modulation alone cannot fix DQN's inherent long-horizon credit assignment failures. Second, an entropy-regularized variant (CtxPipe-Soft)

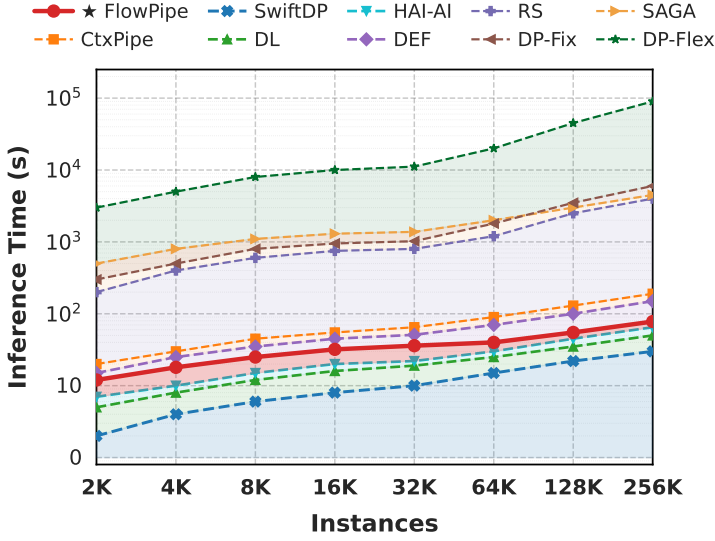


Fig. 6. Scalability under different dataset sizes.

intended to mimic flow-matching paradoxically *degrades* accuracy (0.756 vs. 0.806), as forced localized stochasticity in a fragile operator space frequently triggers fatally compounding invalid actions.

Necessity of Flow-Based Mode Coverage. Disabling reward-proportional sampling in favor of deterministic decoding (w/ Greedy) drops the average accuracy on DiffPrep from 0.896 to 0.886. This confirms that structural mode coverage is essential to bypass local optima and discover the high-reward regions. For instance, on *page-blocks*, FlowPipe robustly explores distinct yet equally optimal paths (InteractionFeatures + Normalizer vs. PCA_AUTO + StandardScaler) before convergence. The top candidates evaluated during this search exhibit a minimal accuracy variance of 0.0012, proving FlowPipe reliably maps high-reward plateaus without collapsing into sub-optimal deterministic traps.

7.5 Scalability and Robustness (RQ4)

Finally, we address **RQ4** by examining the system’s scalability under varying data loads and its robustness to hyperparameter variations.

Row Scalability. We evaluate execution efficiency across 9 validated OpenML datasets, with sizes increasing exponentially from 2K to 256K instances. As plotted on a log-log scale in Figure 6, baselines exhibit distinct performance degradation. Specifically, the gradient-based DP-Flex suffers severe computational overhead, showing an order-of-magnitude runtime spike for datasets exceeding 32K rows due to iterative backpropagation. Similarly, RS and SAGA display linear or super-linear growth driven by repetitive search. In stark contrast, FlowPipe maintains a low and stable inference time despite exponential data growth. This confirms that our amortized inference successfully decouples pipeline generation complexity from dataset volume, ensuring viability for large-scale scenarios.

Feature Scalability. To isolate the impact of feature dimensionality (D), we compare datasets with $\sim 7K$ rows but varying D , from HTRU_2 ($D = 9$) to home_credit ($D = 343$). FlowPipe remains highly robust across this extreme range (0.988 vs. 0.964 accuracy), significantly outperforming the

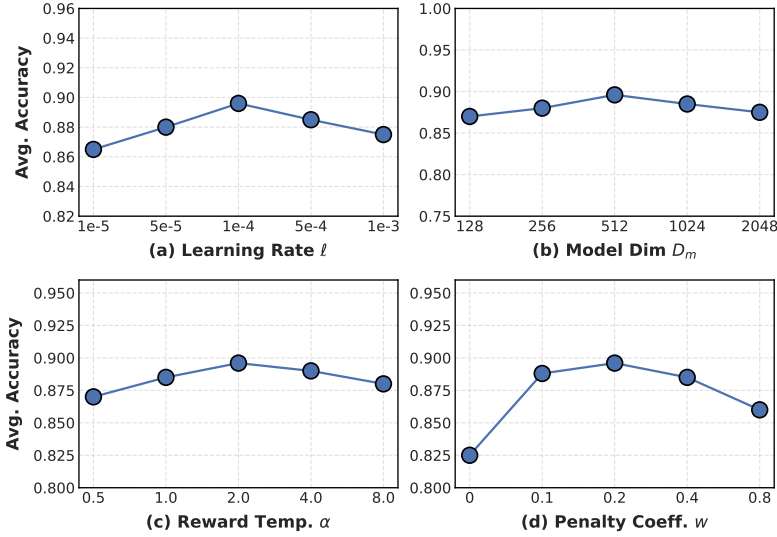


Fig. 7. Hyperparameter sensitivity analysis.

top baseline, CtxPipe (0.951 vs. 0.914). This stability is achieved because mean pooling (Sec. 5.2) rigorously converts variable-length LLM metadata into a fixed-size conditioning vector. Consequently, structural search complexity is natively decoupled from feature width, effectively averting the curse of dimensionality.

Table 6. Scalability of Component Expansion

Space Config.	Performance				Overhead
	Acc	Prec.	Rec.	F1	
FlowPipe	0.896	0.788	0.794	0.780	-
+ Category	0.899	0.794	0.812	0.803	+4.712s
+ Operators	0.898	0.786	0.794	0.790	+0.009s

Component Extensibility. As shown in Table 6, we evaluate FlowPipe’s **extensibility and search space scalability** by expanding the search space with an *Outlier Handling* category and advanced operators (e.g., TargetEncoder). FlowPipe maintains robust performance with minimal latency increases. This scalability stems from our hierarchical policy and FiLM-based gating, which reduce search complexity to $\mathcal{O}(C+K)$ and prune semantically incompatible branches. By leveraging LLM priors to bypass redundant steps, FlowPipe prevents combinatorial explosion regardless of the component pool size.

Hyperparameter Sensitivity. We analyze the impact of key training dynamics: learning rate ℓ , model dimension D_m , reward temperature α , and failure penalty w . As shown in Figure 7, FlowPipe demonstrates robustness across wide ranges, achieving peak performance at $\ell = 1e-4$, $D_m = 512$, $\alpha = 2.0$, and $w = 0.2$. Consequently, minimal tuning is required. Notably, performance shows a stronger positive correlation with D_m . This indicates that a larger model dimension is critical for expanding the capacity of the semantic context embedding, enabling the agent to better assimilate complex reasoning priors from the LLM.

8 Conclusion

This paper presents FlowPipe, bridging generative flow modeling with semantic reasoning to redefine automated data preparation. By synergizing C-GFlowNets, FiLM-based modulation, and failure-aware exploration, it constrains combinatorial search to semantically plausible regions, resolving the inefficiencies of prior RL systems. Across 74 datasets, FlowPipe significantly outperforms SOTA baselines, achieving an **11.96%** accuracy gain and **12.5×** faster training. Future work will explore *dynamic semantic reconditioning* to adaptively capture evolving feature distributions.

Acknowledgments

This work is supported by the Advanced Materials-National Science and Technology Major Project (Grant No. 2024ZD0607700).

Appendix

In the appendix, we report the full evaluation results of FlowPipe on the DiffPrep and DeepLine datasets, supporting the findings in Section 7.2.2. Tables 7 and 8 present the test accuracies, with **bold** and underlined values indicating the best and second-best results, respectively.

FlowPipe achieves the highest accuracy on **14 of 18** DiffPrep datasets and **48 of 56** DeepLine datasets. Its average accuracies are **0.896** and **0.912**, respectively, substantially exceeding the previous SOTA results of 0.806 and 0.813. These results demonstrate FlowPipe’s robustness and its effectiveness in leveraging semantic information for complex, high-dimensional datasets.

Table 7. Characteristics of the DiffPrep dataset collection and comparison of test accuracy.

Dataset	Data Characteristics					Test Accuracy										
	#Inst.	#Feat.	#Lbl.	%Miss.	%Cat.	(ES ⁺)	DEF	RS	DP-Fix	DP-Flex	DL	HAI-AI	SAGA	CtxPipe	SwiftDP	Ours
abalone	4177	9	28	0	12.50	<u>0.287</u>	0.240	0.243	0.238	0.271	0.157	0.260	0.255	<u>0.287</u>	0.274	0.522
ada_prior	4562	15	2	0.14	57.14	<u>0.857</u>	0.848	0.844	0.853	0.846	0.803	0.801	0.833	0.818	0.834	0.920
avila	20867	11	12	0	0	0.916	0.553	0.598	0.652	0.633	0.593	0.630	0.636	0.759	0.556	<u>0.837</u>
connect-4	67557	43	3	0	0	<u>0.792</u>	0.659	0.671	0.726	0.702	0.683	0.775	0.758	0.763	0.661	0.891
eeg	14980	15	2	0	0	<u>0.861</u>	0.589	0.658	0.675	0.683	0.607	0.556	0.658	0.740	0.575	0.885
google	9367	9	2	2.19	50.00	<u>0.672</u>	0.586	0.627	0.631	0.661	0.553	0.550	0.596	0.590	0.569	0.819
house	1460	81	2	6.70	53.75	<u>0.955</u>	0.928	0.938	0.932	0.952	0.771	0.928	0.913	0.818	0.938	0.960
jungle_chess	44819	7	3	0	0	<u>0.861</u>	0.668	0.669	0.680	0.687	0.717	0.760	0.745	<u>0.861</u>	0.676	0.872
micro	20000	21	5	0	0	<u>0.634</u>	0.564	0.579	0.595	0.593	0.613	0.633	0.556	0.605	0.573	0.793
mozilla4	15545	6	2	0	0	<u>0.940</u>	0.855	0.922	0.924	0.927	0.747	0.870	0.932	<u>0.940</u>	0.849	0.966
obesity	2111	17	7	0	50.00	0.927	0.775	0.841	0.891	0.874	0.590	0.768	0.751	0.868	0.870	<u>0.911</u>
page-blocks	5473	11	5	0	0	<u>0.973</u>	0.942	0.959	0.959	<u>0.973</u>	0.940	0.935	0.849	0.965	0.964	0.981
pbcseq	1945	19	2	4.13	22.22	<u>0.866</u>	0.710	0.730	0.728	0.725	0.680	0.733	<u>0.866</u>	0.805	0.679	0.874
pol	15000	49	2	0	0	<u>0.977</u>	0.884	0.879	0.903	0.916	0.873	0.916	0.888	0.949	0.887	0.990
run_or_walk	88588	7	2	0	0	0.990	0.719	0.829	0.903	0.912	0.820	0.915	0.832	0.956	0.715	<u>0.976</u>
shuttle	58000	10	7	0	0	1.000	0.964	0.996	0.998	<u>0.999</u>	0.790	0.951	0.405	1.000	0.965	0.998
uscensus	32561	15	2	0.93	57.14	<u>0.854</u>	0.848	0.840	<u>0.854</u>	0.852	0.813	0.807	0.835	0.845	0.829	0.921
wall-robot-nav	5456	25	4	0	0	<u>0.962</u>	0.697	0.872	0.905	0.913	0.927	0.896	0.841	0.946	0.692	0.978

Table 8. Characteristics of the DeepLine dataset collection and comparison of test accuracy

Dataset	Data Characteristics					Test Accuracy										
	#Inst.	#Feat.	#Lbl.	%Miss.	%Cat. (ES*)	DEF	RS	DP-Fix	DP-Flex	DL	HAI-AI	SAGA	CtxPipe	SwiftDP	Ours	
Accident	7500	14	4	0	0	0.665	0.673	0.674	0.673	0.674	0.598	0.600	0.597	0.692	0.657	0.978
adult	7480	15	2	0	60	0.884	0.849	0.851	0.857	0.855	0.737	0.815	0.843	0.829	0.824	0.918
broadwaymult	285	8	7	1.2	50	0.415	0.298	0.561	0.474	0.526	0.175	0.456	0.474	0.512	0.333	0.749
germangss	400	6	17	0	66.7	0.388	0.238	0.225	0.200	0.188	0.275	0.212	0.213	0.442	0.275	0.536
ar4	107	30	2	0	0	0.895	0.905	0.810	0.857	0.810	0.818	0.773	0.773	0.801	0.909	0.905
bank-full	7368	15	2	0	53.3	0.915	0.886	0.891	0.896	0.899	0.840	0.890	0.896	0.890	0.886	0.948
baseball	1340	17	3	0.1	5.9	0.962	0.937	0.944	0.925	0.925	0.937	0.955	0.948	0.931	0.952	0.979
biodeg	1055	42	2	0	2.4	0.910	0.829	0.806	0.810	0.829	0.834	0.877	0.896	0.871	0.863	0.934
blood-transfusion	748	5	2	0	0	0.812	0.752	0.732	0.785	0.732	0.627	0.713	0.720	0.776	0.767	0.891
bng_cmc	55296	10	3	0	0	0.654	0.520	0.535	0.533	0.537	0.527	0.525	0.522	0.648	0.521	0.878
bodyfat	252	15	2	0	6.7	0.992	0.900	0.900	0.960	0.940	0.784	0.980	1.000	0.941	1.000	1.000
braziltourism	412	9	7	2.6	0	0.815	0.707	0.683	0.707	0.695	0.735	0.771	0.711	0.745	0.735	0.885
breast	699	10	2	0.2	10	0.982	0.971	0.950	0.964	0.986	0.900	0.979	0.971	0.938	0.950	0.989
bureau	7125	302	2	30.6	5.3	0.942	0.928	0.929	0.911	0.924	0.923	0.930	0.922	0.917	0.915	0.965
car	1728	7	4	0	100	0.885	0.852	0.913	0.916	0.919	0.963	0.968	0.974	0.953	0.691	0.990
chatfield_4	235	13	2	0	7.7	0.936	0.915	0.915	0.894	0.872	0.660	0.894	0.872	0.859	0.957	0.923
cmc	1473	10	3	0	0	0.612	0.520	0.565	0.561	0.561	0.502	0.512	0.539	0.598	0.539	0.746
credit	1000	21	2	0	66.7	0.825	0.725	0.740	0.765	0.765	0.690	0.705	0.785	0.780	0.710	0.894
crx	690	16	2	0.6	62.5	0.915	0.899	0.906	0.884	0.920	0.870	0.790	0.877	0.867	0.812	0.921
banknote	1372	5	2	0	0	0.995	0.971	0.967	0.942	0.985	0.851	1.000	0.993	0.970	0.971	1.000
dermatology	365	35	6	0.1	0	0.979	0.959	0.959	0.945	0.973	0.986	0.932	0.973	0.950	0.973	0.986
diggle_table_a2	310	9	2	0	11.1	0.984	0.855	0.952	0.952	0.935	0.871	1.000	1.000	0.970	0.968	1.000
disclosure_z	662	4	2	0	25	0.645	0.561	0.492	0.561	0.598	0.564	0.496	0.504	0.616	0.571	0.771
eucalyptus	736	20	5	3	30	0.712	0.592	0.633	0.571	0.687	0.655	0.608	0.649	0.597	0.635	0.788
Frogs_family	7195	23	4	0	4.3	0.988	0.938	0.945	0.944	0.952	0.905	0.985	0.985	0.917	0.947	0.994
Frogs_gen	7195	23	8	0	4.3	0.988	0.945	0.951	0.950	0.955	0.927	0.976	0.984	0.926	0.953	0.993
Frogs_spec	7195	23	10	0	4.3	0.987	0.954	0.958	0.958	0.959	0.943	0.979	0.984	0.932	0.969	0.993
glass	214	10	6	0	0	0.780	0.571	0.667	0.571	0.643	0.605	0.581	0.837	0.710	0.744	0.807
haberman	306	4	2	0	0	0.795	0.738	0.738	0.754	0.770	0.694	0.645	0.710	0.705	0.677	0.823
home_credit	7250	343	2	19.3	0	0.945	0.928	0.928	0.897	0.922	0.917	0.931	0.922	0.914	0.907	0.964
HTRU_2	7000	9	2	0	0	0.984	0.975	0.973	0.976	0.979	0.973	0.974	0.977	0.951	0.983	0.988
ilpd	583	11	2	0	9.1	0.695	0.733	0.741	0.716	0.716	0.718	0.709	0.744	0.753	0.692	0.832
image_seg	2310	20	7	0	5	0.975	0.885	0.939	0.918	0.931	0.973	0.978	0.981	0.899	0.944	0.981
Indian_Liver	583	11	2	0.1	0	0.695	0.741	0.724	0.707	0.724	0.598	0.684	0.752	0.753	0.692	0.857
Iris	150	5	3	0	20	0.980	0.800	0.733	0.867	0.900	0.900	1.000	1.000	0.945	0.933	1.000
irish	500	6	2	1.1	66.7	0.995	0.990	0.970	1.000	1.000	0.460	0.980	0.980	0.963	0.780	1.000
kc3	458	40	2	0	0	0.935	0.912	0.912	0.901	0.890	0.902	0.859	0.902	0.913	0.891	0.961
kidney	76	7	2	0	42.9	0.920	0.667	0.533	0.600	0.733	0.750	0.875	0.688	0.877	0.688	0.968
LendingClub	7370	53	12	3	22.6	0.680	0.507	0.561	0.543	0.547	0.503	0.565	0.546	0.611	0.579	0.756
magic04	7450	11	2	0	9.1	0.810	0.805	0.823	0.831	0.821	0.787	0.895	0.859	0.845	0.793	0.938
mammographic	961	6	2	2.8	0	0.865	0.823	0.812	0.812	0.818	0.736	0.824	0.829	0.827	0.829	0.913
move_libras	360	91	15	0	0	0.905	0.569	0.639	0.653	0.694	0.583	0.819	0.806	0.887	0.750	0.928
no2	500	8	2	0	12.5	0.775	0.590	0.440	0.610	0.620	0.610	0.710	0.740	0.634	0.500	0.819
phoneme	5404	6	2	0	0	0.905	0.761	0.746	0.753	0.778	0.860	0.897	0.911	0.865	0.736	0.927
plasma_retinol	315	14	2	0	28.6	0.725	0.476	0.476	0.540	0.476	0.444	0.587	0.524	0.627	0.540	0.864
pm10	500	8	2	0	12.5	0.790	0.550	0.550	0.550	0.560	0.500	0.630	0.660	0.627	0.590	0.928
schizo	340	14	2	17.5	21.4	0.745	0.574	0.544	0.632	0.618	0.485	0.588	0.588	0.641	0.677	0.804
Skin_NonSkin	7122	4	2	0	0	0.988	0.923	0.912	0.933	0.929	0.970	0.997	0.999	0.969	0.922	0.999
socmob	1156	6	2	0	83.3	0.955	0.931	0.944	0.922	0.935	0.905	0.922	0.944	0.921	0.914	0.969
solar-flare	1066	13	6	0	23.1	0.730	0.746	0.723	0.742	0.746	0.715	0.776	0.724	0.799	0.738	0.902
broken_machine	7490	59	2	9.4	0	0.550	0.687	0.688	0.688	0.689	0.180	0.687	0.690	0.649	0.688	0.831
triazines	186	61	2	0	1.6	0.650	0.676	0.649	0.568	0.595	0.684	0.737	0.684	0.754	0.711	0.903
veteran	137	8	2	0	12.5	0.795	0.593	0.778	0.667	0.593	0.607	0.821	0.750	0.757	0.607	0.845
weatherAUS	7357	25	2	9.1	24	0.997	0.937	0.971	0.975	0.995	0.783	1.000	1.000	0.964	0.982	0.998
wilt	4839	6	2	0	16.7	0.975	0.944	0.967	0.973	0.977	0.960	0.973	0.977	0.953	0.962	0.991
Wine	178	14	3	0	0	0.989	0.971	0.971	0.971	0.971	0.972	0.916	1.000	0.949	0.972	1.000

References

- [1] Ziawasch Abedjan, Xu Chul, Dong Deng, Raul Castro Fernandez, Ihab F Ilyasl, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting Data Errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment* 9, 12 (2016).
- [2] Arthur Asuncion, David Newman, et al. 2007. UCI machine learning repository.
- [3] Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. 2021. Flow network based generative models for non-iterative diverse candidate generation. *Advances in neural information processing systems* 34 (2021), 27381–27394.
- [4] Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J Hu, Mo Tiwari, and Emmanuel Bengio. 2023. Gflownet foundations. *Journal of Machine Learning Research* 24, 210 (2023), 1–55.
- [5] Laure Berti-Equille. 2019. Learn2clean: Optimizing the sequence of tasks for web data preparation. In *The world wide web conference*. 2580–2586.
- [6] Matthias Boehm, Iulian Antonov, Sebastian Baunsgaard, Mark Dokter, Robert Ginthör, Kevin Innerebner, Florijan Klezin, Stefanie Lindstaedt, Arnab Phani, Benjamin Rath, et al. 2019. SystemDS: A declarative machine learning system for the end-to-end data science lifecycle. *arXiv preprint arXiv:1909.02976* (2019).
- [7] Chengliang Chai, Nan Tang, Ju Fan, and Yuyu Luo. 2023. Demystifying artificial intelligence for data preparation. In *Companion of the 2023 International Conference on Management of Data*. 13–20.
- [8] Jing Chang, Chang Liu, Jinbin Huang, Shuyuan Zheng, Rui Mao, and Jianbin Qin. 2025. ShapleyPipe: Hierarchical Shapley Search for Data Preparation Pipeline Construction. *arXiv preprint arXiv:2510.27168* (2025).
- [9] Sibe Chen, Hanbing Liu, Waiting Jin, Xiangyu Sun, Xiaoyao Feng, Ju Fan, Xiaoyong Du, and Nan Tang. 2024. Chatpipe: Orchestrating data preparation pipelines by optimizing human-chatgpt interactions. In *Companion of the 2024 International Conference on Management of Data*. 484–487.
- [10] Sibe Chen, Nan Tang, Ju Fan, Xuemi Yan, Chengliang Chai, Guoliang Li, and Xiaoyong Du. 2023. Haipipe: Combining human-generated and machine-generated pipelines for data preparation. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.
- [11] Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. 2016. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 international conference on management of data*. 2201–2206.
- [12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186.
- [13] Wei Fan, Kunpeng Liu, Hao Liu, Pengyang Wang, Yong Ge, and Yanjie Fu. 2020. Autofs: Automated feature selection via diversity-aware interactive reinforcement learning. In *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1008–1013.
- [14] Saeed Fathollahzadeh, Essam Mansour, and Matthias Boehm. 2025. CatDB: Data-catalog-guided, LLM-based Generation of Data-centric ML Pipelines. *Proceedings of the VLDB Endowment* 18, 8 (2025), 2639–2652.
- [15] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. *Advances in neural information processing systems* 28 (2015).
- [16] Haotian Gao, Shaofeng Cai, Tien Tuan Anh Dinh, Zhiyong Huang, and Beng Chin Ooi. 2024. CtxPipe: Context-aware Data Preparation Pipeline Construction for Machine Learning. *Proceedings of the ACM on Management of Data* 2, 6 (2024), 1–27.
- [17] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).
- [18] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. 2022. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in neural information processing systems* 35 (2022), 507–520.
- [19] Mazhar Hameed and Felix Naumann. 2020. Data preparation: A survey of commercial tools. *ACM sigmod record* 49, 3 (2020), 18–29.
- [20] Yuval Heffetz, Roman Vainshtein, Gilad Katz, and Lior Rokach. 2020. Deepline: Automl tool for pipelines generation using deep reinforcement learning and hierarchical actions filtering. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 2103–2113.
- [21] Benjamin Hilprecht, Christian Hammacher, Eduardo S Reis, Mohamed Abdelaal, and Carsten Binnig. 2023. Diffml: End-to-end differentiable ML pipelines. In *Proceedings of the Seventh Workshop on Data Management for End-to-End Machine Learning*. 1–7.
- [22] Edward J Hu, Nikolay Malkin, Moksh Jain, Katie E Everett, Alexandros Graikos, and Yoshua Bengio. 2023. GFlowNet-EM for learning compositional latent variable models. In *International Conference on Machine Learning*. PMLR, 13528–13549.
- [23] Moksh Jain, Emmanuel Bengio, Alex Hernandez-Garcia, Jarrod Rector-Brooks, Bonaventure FP Dossou, Chanakya Ajit Ekbote, Jie Fu, Tianyu Zhang, Michael Kilgour, Dinghuai Zhang, et al. 2022. Biological sequence design with gflownets.

- In *International Conference on Machine Learning*. PMLR, 9786–9801.
- [24] Min Soo Kang, Sung Yul Park, Myung-Ae Chung, and Dong-hun Han. 2024. Azure Automated Machine Learning. In *NO-CODE AI: Concepts and Applications in Machine Learning, Visualization, and Cloud Platforms*. World Scientific, 263–282.
 - [25] Udayan Khurana, Horst Samulowitz, and Deepak Turaga. 2018. Feature engineering for predictive modeling using reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.
 - [26] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. 2020. Conservative q-learning for offline reinforcement learning. *Advances in neural information processing systems* 33 (2020), 1179–1191.
 - [27] Erin LeDell, Sebastien Poirier, et al. 2020. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, Vol. 2020. 24.
 - [28] Chonho Lee, Zhaojing Luo, Kee Yuan Ngiam, Meihui Zhang, Kaiping Zheng, Gang Chen, Beng Chin Ooi, and Wei Luen James Yip. 2017. Big healthcare data analytics: Challenges and applications. *Handbook of large-scale distributed computing in smart healthcare* (2017), 11–41.
 - [29] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research* 18, 185 (2018), 1–52.
 - [30] Liangwei Li, Yiyi Zhang, and Ning Wang. 2025. SwiftDP: An Efficient Framework for Automated Data Preparation Pipeline Generation. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE, 4572–4575.
 - [31] Peng Li, Zhiyi Chen, Xu Chu, and Kexin Rong. 2023. Diffprep: Differentiable data preprocessing pipeline search for learning over tabular data. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–26.
 - [32] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2024. Table-gpt: Table fine-tuned gpt for diverse table tasks. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–28.
 - [33] Yonggang Li, Guosheng Hu, Yongtao Wang, Timothy Hospedales, Neil M Robertson, and Yongxin Yang. 2020. Differentiable automatic data augmentation. In *European conference on computer vision*. Springer, 580–595.
 - [34] Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281* (2023).
 - [35] Hanxiao Liu, Karen Simonyan, and Yiming Yang. [n. d.]. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations*.
 - [36] Zhaojing Luo, Sai Ho Yeung, Meihui Zhang, Kaiping Zheng, Lei Zhu, Gang Chen, Feiyi Fan, Qian Lin, Kee Yuan Ngiam, and Beng Chin Ooi. 2021. MLCask: Efficient management of component evolution in collaborative data analytics pipelines. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 1655–1666.
 - [37] Angelo Mozzillo, Luca Zecchini, Luca Gagliardelli, Adeel Aslam, Sonia Bergamaschi, and Giovanni Simonini. 2023. Evaluation of dataframe libraries for data preparation on a single machine. *arXiv preprint arXiv:2312.11122* (2023).
 - [38] Avnika Narayan, Ines Chami, Laurel Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *Proceedings of the VLDB Endowment* 16, 4 (2022), 738–746.
 - [39] Puhua Niu, Shili Wu, Mingzhou Fan, and Xiaoning Qian. 2024. GFlowNet training by policy gradients. In *Proceedings of the 41st International Conference on Machine Learning*. 38344–38380.
 - [40] Randal S Olson and Jason H Moore. 2016. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*. PMLR, 66–74.
 - [41] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. 2018. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.
 - [42] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. [n. d.]. Efficient Neural Architecture Search via Parameter Sharing. ([n. d.]).
 - [43] Neoklis Polyzotis, Martin Zinkevich, Sudip Roy, Eric Breck, and Steven Whang. 2019. Data validation for machine learning. *Proceedings of machine learning and systems* 1 (2019), 334–347.
 - [44] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann, and Andreas Grafberger. 2018. Automating large-scale data quality verification. *Proceedings of the VLDB Endowment* 11, 12 (2018), 1781–1794.
 - [45] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).
 - [46] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. *Advances in neural information processing systems* 28 (2015).
 - [47] Zeyuan Shang, Emanuel Zraggen, Benedetto Buratti, Ferdinand Kossman, Philipp Eichmann, Yeounoh Chung, Carsten Binnig, Eli Upfal, and Tim Kraska. 2019. Democratizing data science through interactive curation of ml pipelines. In *Proceedings of the 2019 international conference on management of data*. 1171–1188.
 - [48] Ravid Shwartz-Ziv and Amitai Armon. 2022. Tabular data: Deep learning is not all you need. *Information Fusion* 81 (2022), 84–90.

- [49] Shafaq Siddiqi, Roman Kern, and Matthias Boehm. 2023. SAGA: A scalable framework for optimizing data cleaning pipelines for machine learning applications. *Proceedings of the ACM on Management of Data* 1, 3 (2023), 1–26.
- [50] Tiago Silva, Rodrigo Barreto Alves, Eliezer de Souza da Silva, Amauri H Souza, Vikas Garg, Samuel Kaski, and Diego Mesquita. 2025. When do GFlowNets learn the right distribution?. In *The Thirteenth International Conference on Learning Representations*.
- [51] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 847–855.
- [52] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 30.
- [53] Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. 2014. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter* 15, 2 (2014), 49–60.
- [54] Shuhei Watanabe. 2023. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance. *arXiv preprint arXiv:2304.11127* (2023).
- [55] Doris Xin, Hui Miao, Aditya Parameswaran, and Neoklis Polyzotis. 2021. Production machine learning pipelines: Empirical analysis and optimization opportunities. In *Proceedings of the 2021 international conference on management of data*. 2639–2652.
- [56] Chengrun Yang, Yuji Akimoto, Dae Won Kim, and Madeleine Udell. 2019. OBOE: Collaborative filtering for AutoML model selection. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 1173–1183.
- [57] Junwen Yang, Yeye He, and Surajit Chaudhuri. 2021. Auto-pipeline: synthesizing complex data pipelines by-target using reinforcement learning and search. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2563–2575.
- [58] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 8413–8426.
- [59] Gyeong-In Yu, Saeed Amizadeh, Sehoon Kim, Artidoro Pagnoni, Ce Zhang, Byung-Gon Chun, Markus Weimer, and Matteo Interlandi. 2021. WindTunnel: towards differentiable ML pipelines beyond a single model. *Proceedings of the VLDB Endowment* 15, 1 (2021), 11–20.
- [60] Ehtisham Zaidi, Rita Sallam, and Shubhangi Vashisth. 2017. Market Guide for Data Preparation. *cit. on* (2017), 2.
- [61] Dinghui Zhang, Hanjun Dai, Nikolay Malkin, Aaron C Courville, Yoshua Bengio, and Ling Pan. 2023. Let the flows tell: Solving graph combinatorial problems with gflownets. *Advances in neural information processing systems* 36 (2023), 11952–11969.
- [62] Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. 2023. Large language models as data preprocessors. *arXiv preprint arXiv:2308.16361* (2023).

Received 17 January 2026; revised 19 May 2026; accepted 12 June 2026