# FlowPilot: A Suggestion System for Designing Scientific Workflows

MAHDI ESMAILOGHLI, Humboldt-Universität zu Berlin, Germany
MATTHIAS WEIDLICH, Humboldt-Universität zu Berlin, Germany

Scientific Workflows (SWF) encapsulate data processing tasks by organizing various tools, operators, and data in a logical flow. Due to their complex and domain-specific nature, developing SWFs remains laborious. Current code-generating large language models (Code LLMs) struggle to assist users in developing these workflows. This limitation arises primarily from the insufficient availability of relevant training data in public repositories, making it challenging for LLMs to learn specialized patterns and domain-specific logic.

To address this, we propose FlowPilot, a suggestion framework for developing SWFs that assists developers by suggesting the next operator. Our system complements Code LLMs by enriching the source code to generate more accurate results. FlowPilot leverages a similarity knowledge base (SKB) that indexes historical workflows to find the ones matching the current context. To generate relevant suggestions, FlowPilot employs a statistical approach based on Markov chains to identify the most likely next step. As a proof of concept, we evaluated our system on NextFlow workflows and the results demonstrate the effectiveness of FlowPilot. It outperforms state-of-the-art code-generating models, e.g., Llama-4, and traditional methods, e.g., association rule mining techniques.

CCS Concepts: • **Information systems → Retrieval tasks and goals**.

Additional Key Words and Phrases: Scientific Workflows, Code Recommender, Retrieval Augmented Generation, Knowledge Discovery

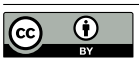
**ACM Reference Format:**
Mahdi Esmailoghli and Matthias Weidlich . 2026. FlowPilot: A Suggestion System for Designing Scientific Workflows. *Proc. ACM Manag. Data* 4, 1 (SIGMOD), Article 39 (February 2026), 27 pages. https://doi.org/10.1145/3786653

## 1 Introduction

Repetitive data analysis in various domains, from bioinformatics to remote sensing, calls for models and methods for automating data processing by the definition of scientific workflows (SWFs) [3, 5, 6]. Despite frameworks for implementing and executing SWFs, such as NextFlow [7] and Texera [45], implementing SWFs remains complex and time-consuming, demanding significant user effort. This complexity is sourced from the need to integrate various tools, dependencies, and data sources. Additionally, ensuring reproducibility, robustness, and code compatibility with workflow standards significantly increases the time and effort required by users. Judging based on GitHub repositories of released NextFlow projects, a.k.a. the NF-Core corpus [11], the average time to develop a workflow is 40 months, involving 14 developers on average.

To reduce development costs, one can leverage a Code-generating large language model (Code LLM), for instance, CoPilot [16] and CodeT5+ [44], to generate suggestions. Code LLMs have

Authors' Contact Information: Mahdi Esmailoghli, Humboldt-Universität zu Berlin, Berlin, Germany, mahdi.esmailoghli@hu-berlin.de; Matthias Weidlich, Humboldt-Universität zu Berlin, Berlin, Germany, matthias.weidlich@hu-berlin.de.




ACM 2836-6573/2026/2-ART39
https://doi.org/10.1145/3786653

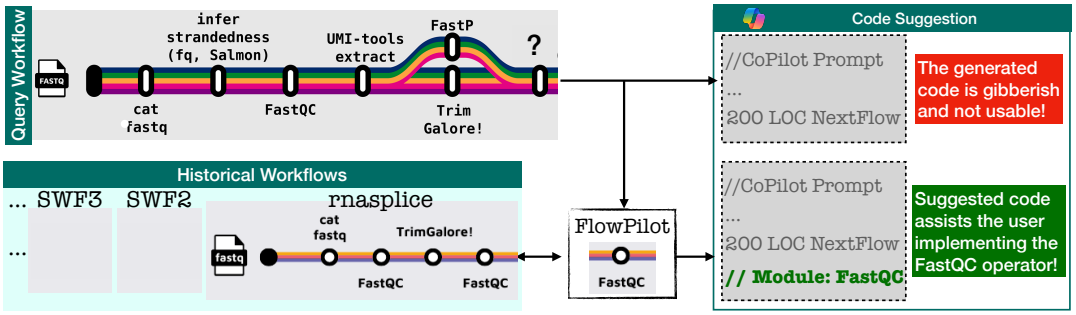



Fig. 1. Illustration of FlowPilot's role in assisting code suggestion.

demonstrated expedition in code development with advanced auto-completion abilities [32, 53]. However, these models are rendered ineffective in the context of SWFs for several reasons [36, 40]:

(1) **Limited training data.** Workflow repositories compose a negligible ratio of publicly available code bases on which generative models are trained. This limitation in code bases makes Code LLMs less effective in suggesting corresponding lines of code for scientific workflows. In particular, there are fewer than 5000 NextFlow repositories on GitHub out of over $420M$. This is 0.00001 of all repositories. The prevalence of general-purpose source code, as opposed to domain-specific workflows, makes code generators perform well in solving common boilerplate problems, but poorly on workflows involving intricate computations on complex data [21, 22, 48].

(2) **Data- vs. control-flow development.** Code completion models are optimized for control flow and sequential line-by-line code generation, but SWFs are driven by data flow. The development of SWFs depends on selecting and composing modules based on semantic data dependencies, not step-by-step logic. As a result, without adaptation, Code LLMs are poorly suited for building SWFs [21].

(3) **Lack of explicit objective.** Code LLMs generate predictions based on localized syntactic cues present in the code, such as function signatures and inline comments. In contrast, effective suggestions for SWFs require reasoning over the global semantics of existing modules and an understanding of the workflow's scientific objective, information that is rarely made explicit in the code.

To overcome these issues and to provide effective support in the design of SWFs, two main challenges must be addressed:

**Contextual relevance.** Scientific workflows are inherently domain-specific, with each domain exhibiting unique context, structures, and operator usage patterns. A major challenge for supporting the design of a workflow is to accurately identify its domain context, often from partial or heterogeneous information. Even within the correct domain, generating relevant suggestions demands an understanding of the workflow's current state and objective. This involves interpreting not just the sequence of existing operators but also the underlying intent, requiring domain-aware reasoning.

**Operational relevance.** Support for workflow design must be efficient, especially in interactive environments. The reason is that generating suggestions involves retrieving and evaluating a potentially vast number of candidates.

In this paper, we tackle the above challenges with FlowPilot, a suggestion system for scientific workflows. It leverages historical workflows, stored in public corpora, such as GitHub, to identify the next operator to be implemented in a given incomplete workflow. As such, it enhances generative

models by injecting relevant domain knowledge into a Code LLM's suggestion process. We illustrate the use of FlowPilot by means of the following example.

*Example.* In Figure 1, a user designs a workflow to analyze RNA sequencing data [33]. The upper left box visualizes the workflow stub, which, in practice, contains around 200 lines of Nextflow code. Using this stub as a query for a Code LLM is not meaningful, as it generates dozens of lines of unrelated Nextflow code, mostly comprising generic comments. While the code is syntactically correct, it is useless for workflow design, as the Code LLM is not aware of the semantics of the workflow to be developed.
This issue is addressed with FlowPilot. In this example, FlowPilot realizes that there is a similar historical workflow [1] that implements operators `cat fastq`, `FastQC`, `Trim Galore!`, and `FastQC` in succession, which suggests that the next operator to be implemented must be `FastQC` for data quality control. Once FlowPilot identifies the next operator, it may be incorporated, e.g., by injecting a comment in the code, and a Code LLM can be prompted to suggest the correct code completion.

At a technical level, we make several contributions:

(1) We propose a solution to identify workflow domains. We extract informative features from workflows and generate searchable domain profiles. These profiles are used to pinpoint relevant sub-workflows, from which we generate suggestions.

(2) To enable efficient access to relevant domains, sub-workflows, and ultimately suggestions, we construct a Similarity Knowledge Base (SKB). It contains structural, domain, and semantic details of historical SWFs. Furthermore, we build a hierarchical index structure on top of the SKB for efficient retrieval.

(3) Given an SKB of relevant historical workflows, we propose a suggestion engine using Markov chains to effectively discover the next operator.

(4) To incorporate user feedback and to remain effective in unforeseen domains, we introduce an online learning strategy that learns from user feedback by updating the SKB.

We assess our system's performance by comparing it against state-of-the-art models and established baselines. The experimental results show that FlowPilot significantly outperforms computationally intensive baselines across a range of metrics.

## 2 The Problem of SWF Suggestion

In this work, we target next operator (also, tool or module) suggestion for SWFs: *given an incomplete workflow $Q$, referred to as the query workflow, and a corpus of SWFs $W$, the goal is to find the correct operator from $W$ that complements $Q$.*

Following the traditional Retrieval Augmented Generation (RAG) paradigm [19], which involves embedding, indexing for retrieval, and response generation, we break the above problem into three sub-problems: domain discovery, which concerns relevant embeddings, the construction of a Similarity Knowledge Base (SKB), which covers retrieval aspect of the pipeline, and suggestion generation, which corresponds to generating the final output.

### 2.1 Domain Discovery

As a first step, we aim at identifying workflows that are relevant to $Q$, i.e., a subset of $W$ that facilitates meaningful and precise suggestions. We call this relevant subset the domain of $Q$, denoted $D_Q$. Note that $D_Q$ is dynamic and changes as the SWF is developed. For instance, the first $D_Q$ can refer to a pre-processing step in an RNA sequence analysis workflow; continuing the development of the workflow, the domain changes to processing and post-processing of the sequenced data.

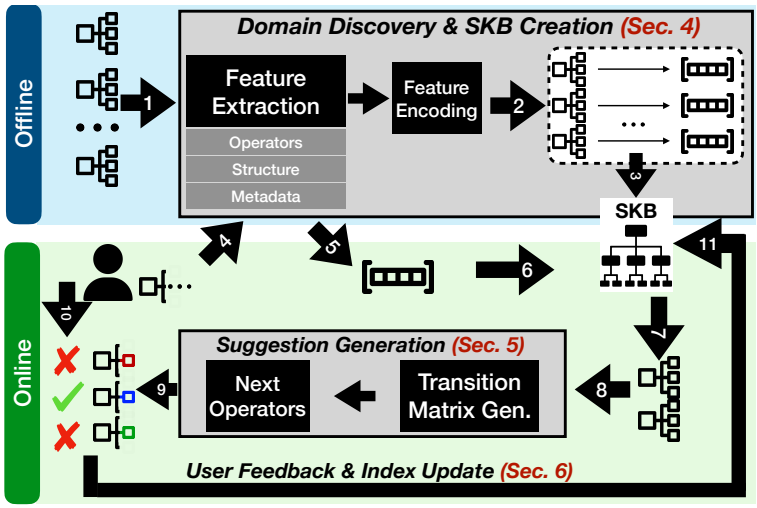



Fig. 2. FlowPilot's architecture.

Addressing the domain discovery problem for an incomplete SWF requires answering the following questions:

- *How to define a domain of a workflow and what features constitute the domain definition and discovery of that workflow?*
- *How to split workflows in order to differentiate multiple semantics and domains within each SWF?*

### 2.2 SKB Creation

Next, we need to construct a data structure to obtain the relevant domains, based on which we generate suggestions. Domain discovery is the problem of defining domains and corresponding features, while SKB creation refers to making these features searchable. This involves answering the following questions:

- *How to featurize workflows in $W$ to facilitate finer-grained (sub-workflow-level) domain discovery?*
- *How to build the SKB to enable efficient search among a large number of possible domains?*

### 2.3 Suggestion generation

Given a set of top-k candidate workflows $C \subseteq W$ with $|C| \leq k$, the goal is to identify the correct operator from $c \in C$ to extend $Q$ by one operator. To this end, we answer the following questions:

- *Which operator from the identified candidate workflow is most likely to correctly extend the query workflow?*
- *How to incorporate user feedback on the selected operator to generate better suggestions in the future?*

## 3 Overview of FlowPilot

To address next operator suggestion for SWFs, we present FlowPilot, as summarized in Figure 2. FlowPilot involves two phases: an offline phase, in which a corpus of existing SWFs is analyzed and a Similarity Knowledge Base (SKB) is constructed; and an online phase, in which the system generates new suggestions, enabling a user to continue developing the SWF at hand.

### 3.1 Offline Phase

To construct the SKB from a given set of historical SWFs, each workflow is broken into sequential sub-workflows. To capture the domain of each sub-workflow, we extract syntactic and semantic features. Specifically, we consider operator features, indicating the operators involved in each sub-workflow; structure information, representing the connection among operators; and metadata, including high-level information regarding the application of the workflow (Step 1). Once these features have been extracted, they are encoded into a high-dimensional space (Step 2). Each sub-workflow is mapped to a vector capturing the extracted features, i.e., the domain. To execute search queries efficiently, we index the obtained vector embeddings using an HNSW index structure [12] (Step 3). This index serves as the SKB for our suggestion system. Section 4 describes this offline phase in more detail.

### 3.2 Online Phase

Using the SKB, FlowPilot can generate suggestions for an incomplete, i.e., query, workflow. Similar to the offline phase, the query workflow is featurized (Step 4) and its embedding, i.e., domain representation, is generated (Step 5). FlowPilot then searches the SKB for the most similar domains (Step 6). This way, the $K$-nearest neighbors with minimum distance to the domain of the query workflow are identified (Step 7). Using the top-$K$ similar domains, FlowPilot fetches the corresponding sub-workflows and builds a Markov model to predict the next operator (Step 8). To this end, we generate a transition matrix, indicating the likelihood of each operator being the next operator for the query workflow. The operators with the highest probability are chosen to be suggested to the user (Step 9). The process of suggestion generation based on the Markov model is explained in Section 5.

A user may choose the most relevant operator from the suggestions and continue designing the workflow (Step 10). Then, the new workflow is considered the query workflow to generate the next suggestion. Moreover, the system utilizes the user's feedback, which is generated by choosing the correct suggestion, to update the SKB accordingly (Step 11). This benefits the system in multiple ways: first, it allows the system to capture each user's personal preferences, e.g., on operators or tools with which the user is particularly familiar. Second, learning from the query workflow benefits the design of future workflows, e.g., by learning about operators that have not been used before. Third, the system learns to avoid making similar mistakes repeatedly. By updating the index with user feedback, more weight is given to correct operators. We describe this process of updating the SKB in Section 6.

## 4 Domain Discovery and SKB Creation

In this section, we address the problem of discovering and materializing domains for SWFs. First, we define the concept of a domain and its granularity. Then, we elaborate on features contributing to identifying domains. Finally, we discuss how we extract these features from workflows and index them for efficient retrieval.

### 4.1 Domains and their Granularity

Intuitively, the domain of a workflow represents the specific area of application it serves. This ranges from coarse-grained areas like bioinformatics or materials science to more fine-grained specializations, such as particular types of DNA analysis (e.g., nf-core/sarek for somatic and germline mutation calling) or RNA sequencing (e.g., nf-core/rnaseq for transcriptome analysis). Domains may also be assigned to sub-workflows, which is essential for large-scale workflows that encompass

various functionalities. In any case, to generate relevant suggestions, it is crucial to identify the domain of the query (sub-)workflow accurately.

*Example. rnaseq* [33] and *rnasplice* [1] are two NF-Core SWFs. Although both process RNA sequence data, they differ in their objectives. The former analyzes RNA data and produces a gene expression matrix and a quality control report. The latter is a pipeline for alternative splicing analysis of this data. Regardless of different objectives, they are similar in the first stage, namely, pre-processing (Figure 1). In particular, both use `cat` to merge *FastQ* files, use `Trim Galore` to apply adapter and quality trimming, and leverage `FastQC` as the quality control tool.

In this example, one can use the first two stages of one SWF to generate suggestions for developing the other. This emphasizes the importance of finding relevant domains for suggestions.

### 4.2 (Sub)Workflow Elements

To extract the domain of a sub-workflow, we analyze the elements that define it. Unlike programming code, which primarily expresses domain through control-flow, function logic, and library usage, scientific workflows emphasize data flow and how data moves and transforms through defined steps. Hence, domain discovery for scientific workflows shall include the operators involved in the workflow, intermediate data between the operators, the workflow structure, its input and output, its description, and additional parameters (workflow schema) used in the workflow definition.

*4.2.1 Operators.* We include the set of operators, as workflows within the same domain commonly use similar operators. For instance, bioinformatics workflows tend to use operators provided by the BAM tool to process genomic data, such as alignment, sorting, and indexing of DNA sequences. This statement extends to finer-grained domains of sub-workflows.

*4.2.2 Structure.* We include the order in which operators are connected by means of the *"Directly Follows"* and the *"Eventually Follows"* relations. They indicate if one operator directly succeeds another or appears at any later point in the sequence, respectively. If operator $A$ directly follows operator $B$, it means $A$ occurs immediately after $B$ in the sequence. In contrast, if $A$ eventually follows $B$, $A$ appears at any further point after $B$, though not necessarily immediately. These structural relations capture crucial information about the semantic dependency of operators. For example, quality control (FastQC) directly follows raw sequencing input, while alignment (STAR) eventually follows pre-processing, leading to reproducible and efficient high-throughput data analysis.

*4.2.3 Intermediate data channels.* Each pair of operators is connected through a data channel, passing data from one operator to the next. These data channels express how directly-connected operators communicate. Furthermore, the intermediate data encapsulates operator semantics regarding the type of functionalities conducted on the previous data. For example, in an NF-Core pipeline, FASTP trims raw reads, producing cleaned FASTQ files that are passed via a data channel to BWA for alignment. This channel not only transfers data but also reflects the transformation; here, raw reads are filtered and trimmed before alignment.

*4.2.4 Workflow description.* If available, workflow descriptions illustrate the objective of the workflow in natural language, which can support domain discovery. Workflow descriptions may highlight similarities and differences between pipelines. For example, as mentioned above, the `rnaseq` and `rnasplice` pipelines both focus on RNA sequencing, but have different objectives. Still, their descriptions point to similar pre-processing and alignment steps.

**Algorithm 1:** Pseudocode for FlowPilot's offline phase.

```
1 Inputs: W: corpus of workflows
2 Output: SKB: semantic knowledge base
3 embeddings ← ∅
4 for w in W do
5 |   all_ngrams ← extractNGrams(w)
6 |   metadata ← extractMetadata(w)
7 |   embeddings ← encode_features(all_ngrams, metadata)
8 SKB ← index(embeddings)
9 return SKB
```

*4.2.5 Workflow schema.* This feature represents parameters per operator. These parameters can adjust the functionality of the system depending on the user's preferences. The schema also contains information regarding the input/output data of the workflow.

### 4.3 Domain Features Definition

We categorize the aforementioned elements into two feature classes: n-gram-level features and workflow-level features.

*4.3.1 N-gram-level features.* Features that are defined for a workflow as a whole may be too coarse-grained to support effective suggestions. To pinpoint the sub-workflows that are most relevant to the specific part of a workflow that is developed by a user when a suggestion shall be made, features need to be extracted at a finer granularity. Features at the operator level, the smallest unit in workflows, would not be meaningful, as they do not capture the workflow structure. Therefore, to incorporate intermediate data and the workflow structure, we consider features at the level of n-grams, i.e., paths of size $n$, from the Directed Acyclic Graph (DAG) representation of workflows, as follows.

A DAG representation of a workflow $w$ is a directed, edge-labeled, acyclic graph $G_w = (V_w, E_w, d_w)$. Here, $V_w$ is a set of operators that represent nodes of the graph, $E_w \subseteq V_w \times V_w$ models the data flow, and $d_w : V_w \rightarrow D_w$ assigns input data elements to each operator, with $D_w$ denoting all possible data elements. This way, the output data elements of each operator are implicitly induced by the data input of the succeeding operator in the workflow DAG.

Unlike the state of the art [46], an n-gram does not only capture the operators involved in a workflow, but also the data passed among those operators. Each n-gram corresponds to a path in the workflow's DAG and is represented as a sequence of data elements and operators: $G_n = d_w(o_1), o_1, d_w(o_2), o_2, \ldots, [d(o_n), o_n]$, where $(o_i, o_{i+1}) \in E_w$ for $1 \leq i < n$. Using n-grams, the operators, their intermediate data, and the workflow structure are captured.

*4.3.2 Workflow-level metadata features.* Workflow description, input/output data, and schema information are workflow-level features. Thus, this information is extracted per workflow.

### 4.4 Feature Extraction and Indexing

Having defined the features, we now turn to their extraction and indexing, which corresponds to the offline phase of FlowPilot from Figure 2. As shown in Algorithm 1, this phase takes a set of historical workflows as input (Line 1) to construct the Similarity Knowledge Base (SKB). For each workflow, we extract n-gram-level features and workflow-level metadata features, and encode

**Algorithm 2:** Pseudocode for FlowPilot's online phase.

```
1  Inputs: SKB: similarity knowledge base, K: the number of retrieved nearest neighbours, θ:
     certainty threshold, R: the number of suggestions, W_q: query SWF, N_Q: the active n-gram
2  Output: R suggestions
3  metadata_q ← extractMetadata(w_q)
4  feature_q ← encode_features(N_Q, metadata_q)
5  KNN, distances ← SKB.query(feature_q, K)
6  model ← train_Markov_model(KNN, distances)
7  suggestions, certainty ← model.query(N_Q, R)
8  correct_operator ← GetUserFeedback(suggestions)
9  if certainty < θ then
10 |   SKB.add(N_Q + correct_operator)
11 else if correct_operator NOT IN suggestions then
12 |   SKB.delete(KNN)
13 |   SKB.add(N_Q + correct_operator)
```

them as embeddings (Lines 5 - 7). These embeddings are indexed and returned for querying in the online phase (Lines 8 and 9).

*4.4.1 N-gram encoding.* The encodings of n-grams should not only capture the operators individually but also their relations with input and output data and neighboring operators. Therefore, standard embedding approaches, such as bag-of-words, are ineffective as they ignore contextual dependencies. As such, they fail to preserve the syntactic structure and semantic relations that are essential for meaningful n-gram representations. Also, these embeddings do not incorporate the data/operator and operator/operator relations.

To address these issues, we transform each n-gram into textual data by concatenating the corresponding operator and data elements. Each sentence not only incorporates the operator and data dependencies but also encodes operations done on data elements, capturing semantics and the domain of the n-gram. For instance, "ch_multiqc_files Samtools_Index" expresses that the data for *multiqc* files is indexed using the *Samtool*'s indexing operator. Based thereon, we obtain embeddings with a sentence transformer [20].

*4.4.2 Metadata Encoding.* Workflow features, including the description, schema, and the input/output definition of the workflow, are normally stored as a text file as part of the workflow definition. Hence, similar to n-grams, we use a text-to-embedding transformation, where each feature is embedded independently, and the results are averaged. The metadata embedding is concatenated with the n-gram embeddings of the workflow to generate the final data point corresponding to the domain of the n-gram.

*4.4.3 Indexing SKB.* Obtaining all embeddings for the historical data, we index them for efficient K-Nearest Neighbors (KNN) search. To this end, we employ the HNSW (Hierarchical Navigable Small World) index [29], which is efficient for the evaluation of KNN queries on high-dimensional vectors [12].

## 5 The Suggestion Generator

In this section, we provide details on our generator for next-operator suggestions. Algorithm 2 illustrates the general process of the online phase of FlowPilot. Its input consists of the Similarity

Knowledge Base (SKB) and several configuration parameters. It is applied for a given query workflow, for which FlowPilot extracts the active n-gram, i.e., the n-gram that the developer currently implements and for which the next-operator suggestions shall be returned. We refer to this n-gram as the query n-gram, denoted as $N_Q$. To generate the suggestions, FlowPilot constructs a Markov chain model using the operators in the $K$-nearest neighbors n-grams (Lines 3 - 6). This model is then used to predict the most probable next operator in the query workflow (Line 7). Afterwards, user feedback is incorporated as detailed in Section 6 (Lines 8 - 13).

Below, we provide further details on the derivation of the $K$-nearest neighbors n-grams, the selection of operators from these n-grams to be considered for suggestions, the idea behind using a Markov chain model for suggestions, as well as its construction.

### 5.1 Obtaining the KNN

We adopt the paradigm of retrieval-augmented generation (RAG), where approximate nearest-neighbor search is a common technique to obtain relevant candidate suggestions. This step is in particular crucial in our use case, since obtaining the KNN n-grams provides the relevant domain for the SWF that is being developed. Limiting the search to only top-K similar n-grams not only reduces the search space for efficient suggestion generation, but also shields the system from noisy instances, rendering it more robust.

In general, we use $N_Q$ to find its KNN n-grams. Similar to the feature extraction step for historical data, we generate the embedding corresponding to the query n-gram. Then, this embedding is queried against the SKB and top-$K$ similar embeddings and their corresponding n-grams are retrieved, represented as $C_Q = \{c_i \mid 1 \leq i \leq K\}$. Although we use operators and data elements in n-grams to generate the SKB and query for the KNN paths, to generate the suggestion, we only need operators. Therefore, each n-gram $c = d_w(o_1), o_1, d_w(o_2), o_2, \ldots, [d(o_n), o_n] \in C_Q$ is simplified into the sequence $c' = o_1, o_2, \ldots, [o_{n-1}, o_n]$ of the contained $n$ (or $n-1$ if the n-gram ended with a data element) operators.

### 5.2 Operator Selection Criteria

Given the set of relevant n-grams ($C_Q$), we establish three criteria that are essential for predicting the next operator.

*5.2.1 Directly or Eventually Follows relations.* These relations indicate whether an operator requires the direct output of another to be executed. Unlike *Eventually Follows* relation, in the case of *Directly Follows* relation, there is no intermediate operator involved. For instance if $A \rightarrow B \rightarrow C$, $B$ *Directly Follows* $A$. Although $C$ follows $A$ at some point, $A$ and $C$ do not have a *Directly Follows* relation, instead, $C$ *Eventually Follows* $A$.

*Directly Follows* and *Eventually Follows* relations between two operators in any $c \in C_Q$ signify that these operators co-occur in at least one n-gram, implying a possible semantic sequence. This semantic connection between the two operators is stronger in the case of *Directly Follows* compared to *Eventually Follows* relations. In fact, we argue that the closer two operators are, the higher the likelihood that they are semantically related.

*Example.* As depicted in Figure 1, the operator `FastQC` *Directly Follows* `Trim_Galore`, as FastQC is often executed immediately after trimming reads to assess quality. On the other hand, `MultiQC` *Eventually Follows* `Trim_Galore`, as it aggregates quality control reports from multiple steps, including `FastQC` and other processes, making the connection more indirect.

*5.2.2 Commutative relation.* Workflows may contain independent operators that can be executed in any order, regardless of their order observed in the DAG representation of the workflow. That is,

their ordering depends only on the user's choice when designing the workflow. To cater for these operators, we also incorporate a relation for operators that can occur in any order.

*Example.* In our running example, the operators `SAMtools_sort` and `MarkDuplicates` may exhibit a commutative relation if their outputs are not dependent on each other's order of execution. Depending on the workflow structure, a developer might choose to sort BAM files before marking duplicates or vice versa, without affecting the overall outcome.

*5.2.3 Similarity of n-grams.* The degree of similarity between the query n-gram and its $K$ closest matches retrieved through the KNN search is not identical. While the index search returns the top-$K$ most similar n-grams, their relevance to $N_Q$ varies. Thus, querying the SKB for the most relevant n-grams, we compute a normalized distance between $N_Q$ and each of its $K$ nearest neighbors. Intuitively, the smaller the distance, the higher the likelihood that the n-gram is relevant to the task at hand. Consequently, we use this distance as a measure of importance to rank similar n-grams.

*Example.* In our example, if the query n-gram contains the operator `Trim_Galore`, two of its closest matches might include `FastQC` and `CutAdapt`, as they often co-occur with `Trim_Galore` in the pre-processing step of workflows. However, their similarity scores will vary, with `CutAdapt` likely being closer to `Trim_Galore` due to their shared role in adapter trimming, whereas `FastQC`, though relevant, serves a distinct quality control function with a slightly higher distance from the query n-gram.

Next, we discuss how to find the next operator among KNN based on the aforementioned criteria.

### 5.3 Suggestion using Markov Chains

To find the most probable next operator for a given $N_Q$ based on its corresponding KNN, we model next-operator prediction as a Markov chain. A Markov chain captures a system that transitions from one state to another within a finite set of states. Transition probabilities govern the likelihood of moving between states.

The benefit of using a Markov model in our context is threefold: *i*) Markov models naturally capture the contextualized probabilities that stem from complex workflow histories. As such, they facilitate traceability and explainability of the suggestions at a much higher level than LSTMs or transformers. *ii*) Markov models do not require large datasets for training, which are scarce in the field of scientific workflows. *iii*) Markov models enable efficient inference, which is crucial when realizing interactive suggestions in SWF development.

Note that the statelessness of Markov models, which contributes to their efficiency and explainability, is covered by the availability of KNN n-grams and operator selection criteria discussed earlier in this section. Using KNN n-grams to train the Markov model, we already ensure that the generated suggestion is highly relevant to the current state of the developed workflow.

To formulate our problem as a Markov chain, we first introduce the states. We consider the cumulative set of operators within $N_Q$ and all the fetched KNN n-grams as the set of states. The transitions between states are derived from observed transitions within the KNN n-grams. These historical transitions in KNN n-grams and the relationships between the operators allow us to calculate the transition probability matrix. The probability matrix represents the likelihood of transitioning from one operator to another, conditioned on the current operator. Based thereon, we can predict the most probable subsequent operator(s) for the query n-gram.

### 5.4 Probability Estimation

Now, we turn to the estimation of the transition probability matrix. Let us define $S$ as the set of operators in all $K + 1$ n-grams, which are $K$ nearest neighbors n-grams fetched from the SKB and

one query n-gram, i.e., $N_Q$. The transition probability matrix is a two-dimensional matrix $M$ with $|S|$ rows and $|S|$ columns, each representing a specific operator. $M[i][j]$ represents the probability of transitioning from state $i$ to state $j$. In the context of next-operator prediction, $M[i][j]$ indicates the probability of the next operator being $s_j$ given that the last operator developed by the user in $N_Q$ is $s_i$. Note that to predict the next operator, we only consider the last operator in $N_Q$. Previous operators are employed to fetch the most relevant n-grams from the SKB; thus, they only impact the prediction indirectly and through $K$ similar n-grams. This is because the predicted operator will *Directly Follow* the last operator.

To create the transition matrix, first, we generate a frequency matrix ($F$) with the same dimensions as $M$. Unlike $M$, $F$ does not guarantee that the cumulative probabilities in the $i^{th}$ row of the matrix are equal to 1. We initialize $F$ by $F[i][j] = \epsilon$, where $\epsilon \ll \frac{1}{|S|}$, i.e., $\epsilon$ is very small. This is done to prevent dead-end transitions where the probability of moving to any next state from the current state is zero. Intuitively, in scenarios where the operators in n-grams do not co-occur with the last operator in $N_Q$, we consider a uniform chance of any other operator to be predicted.

Inspired by our discussion in Section 5.2, we define three variables that contribute to the frequency matrix: $x()$, $y()$, and $z()$. Assume $c_i$ is the $i^{th}$ n-gram in the KNN set of n-grams and $d_{c_i}$ is a normalized Euclidean distance between $c_i$ and $N_Q$ in the SKB, where $0 \leq d_{c_i} \leq 1$. Now we define $x()$, $y()$, and $z()$.

$x(A, B)$ captures *Directly* or *Eventually Follows* relations. If $B$ occurs after $A$ in $c_i$, i.e., $B$ follows $A$, then, $x(A, B) > 0$. As we mentioned before, the *Directly Follows* relation is stronger than the *Eventually Follows* relation. Transitively, among two *Eventually Follows* relations, the one with fewer intermediate operators is stronger. To capture this property, we define $x()$ as follows:

$$x(A, B) = \frac{1}{I(B, c_i) - I(A, c_i)},$$

where $I(B, c_i)$ returns the position index of operator $B$ in n-gram $c_i$. The denominator captures the distance between two operators in the n-gram. Based on the definition above, $x()$ is maximal ($x(A, B) = 1$), when $B$ directly follows $A$.

$y(A, B)$ is similar to $x()$, capturing *Directly* or *Eventually Follows* relations but in a reverse direction. If $A$ follows $B$, then $y(A, B) > 0$. This variable is defined to capture the relation between operators that are not necessarily applied sequentially. We define $y()$ as:

$$y(A, B) = \frac{1}{I(A, c_i) - I(B, c_i)}.$$

Note that $x()$ and $y()$ are mutually exclusive:

$$x() \begin{cases} > 0 & \text{if } j > i \\ = 0 & \text{if } i < j \end{cases}, \quad y() \begin{cases} > 0 & \text{if } i < j \\ = 0 & \text{if } j > i \end{cases}.$$

$z(d_{c_i})$ incorporates the distance between $N_Q$ and $c_i$ by emphasizing n-grams with smaller distance to the query n-gram. Specifically, $z()$ increases as the similarity between two n-grams grows, achieving the maximum value ($z(d_{c_i}) = 1$) when the n-grams are identical. We define $z()$ as follows:

$$z(d_p) = 1 - d_{c_i}$$

.

Defining $x()$, $y()$, and $z()$, we fill the frequency matrix as follows:

$$F[A][B] = \epsilon + \sum_{c_i \in C} \cdot \sum_{(A,B) \in c_i \times c_i} (\alpha \cdot x(A, B) + \beta \cdot y(A, B)) \times \gamma \cdot z(d_{c_i}),$$

Table 1. The KB updating rules based on prediction correctness and model uncertainty.

| | **Certain** | **Uncertain** |
|---|---|---|
| **Correct** | ignore | add |
| **Incorrect** | delete & add | add |

where $c_i \times c_i$ is the set of all ordered pairs of operators from $c_i$.

Since the importance of $x()$, $y()$, and $z()$ in calculating the relevance of two operators is not equal, we introduce $\alpha$, $\beta$, and $\gamma$ parameters respectively, as the weights of these scoring functions. We use Bayesian optimization to find the best values for these constants. Using the frequency matrix, $M$ is obtained by normalizing $F$ by dividing each element in $F$ ($F[A][B]$) by the sum of all elements in their corresponding row ($A$):

$$M[A][B] = \frac{F[A][B]}{\sum_{0 \leq s < |S|} F[A][s]}.$$

Based on the transition probability matrix $M$, we generate suggestions. If the last operator in $N_Q$ corresponds to $s_i \in S$, we compare the probability values in row $M[s_i]$ and generate suggestion(s) by choosing the operator(s) corresponding to the maximum probability. Depending on the aforementioned configuration parameters, FlowPilot may provide more than one suggestion, so that the user can select the one that fits best to their needs.

## 6 User Feedback and Index Update

In Sections 4 and 5, we discussed how to create the SKB, look up the KNN n-grams, and generate suggestions for developing SWFs. The SKB is effective in finding relevant n-grams if the indexed corpus contains those n-grams, meaning that the query workflow has similar domains to those of the historical data. If the domain of the query workflow has not been seen before, our suggestion generator will not be able to suggest correct operators. This is because the next correct operator does not exist in the SKB. To address this problem, FlowPilot employs an online-learning process for the SKB using the newly developed n-grams, as included in the last part of Algorithm 2. This way, the system learns the new operators as well as how they are used in the new domain. In this section, we first explain how user feedback is obtained (Line 8), before we outline how user labels enables updates to the SKB (Lines 9 - 13).

### 6.1 User Feedback and Labeling

Traditional active learning approaches aim at minimizing the user labels due to the reason that user involvement is expensive and the ultimate goal is to achieve high accuracy with a minimum number of user labels. Unlike traditional active learning approaches, the user feedback in our suggestion generator is a byproduct of the user developing the SWF by choosing or implementing the correct next operator. In this case, the user continues to develop the workflow regardless of the suggestions, consequently, the system continuously receives user feedback. Therefore, we may compare the generated suggestions to the operators chosen by the user at each step of the development phase. Knowing this, our goal in updating the SKB is not to reduce user labels, but rather, to update the SKB to achieve accurate suggestions for developing SWFs.

FlowPilot incorporates SKB updates by evaluating individual user labels based on two aspects: first, system's confidence in the generated suggestion and second, the alignment of generated suggestions with the user labels.

### 6.2 Updating the SKB

Table 1 depicts the rules FlowPilot uses to update the SKB. Intuitively, if the system is confident in the suggestion and the suggestion made by the system is confirmed to be correct, there is no need to update the SKB. This is because the KNN led to n-grams containing the correct operator, and also, the relations among operators in the KNNs resulted in a transition matrix with a probability distribution that strongly generates the correct suggestion.

On the contrary, if the suggestion for the next operator is proved to be wrong based on the user feedback, while the system is certain about the generated suggestion, the SKB must be updated. High confidence in a wrong suggestion is an indication that the system has over-fitted to irrelevant patterns in the KNN. This overconfidence requires correction by updating the SKB and removing these misleading elements. To do this, first, the correct n-gram, i.e., $N_Q$ followed by the user-provided operator, i.e., the ground truth, is added to the SKB to update the index with the correct suggestion. Then, the n-grams in the KNN resulting in the incorrect prediction are removed from the SKB. The intuition behind the deletion is that the semantics of n-grams do not hold in the new domain.

If the model is not confident about the generated suggestions, the low certainty signals that the KNN is not strongly relevant to the task, indicating that the system is under-fitted. That is, the system requires more labels for this specific domain to reinforce the transition probabilities to make more confident decisions. In this case, the SKB is updated only by adding the correct n-gram, without removing potentially useful but less relevant KNNs. This update is conducted regardless of the correctness of the suggestions.

Note that the added n-grams generated based on the user feedback have identical workflow-related features. This property is desired as it ensures that n-grams generated from the query workflow are prioritized, allowing for the most relevant suggestions to be generated for the current user.

### 6.3 Certainty Calculation for Suggestions

Recall that given $N_Q$ and its KNN n-grams, the system generates the transition probability matrix ($M$). Then, based on the probability distribution of this matrix, in particular the transition from the last developed operator in $N_Q$, FlowPilot suggests the next operator(s). If $s_i$ is the state corresponding to the last operator in $N_Q$, $M[s_i]$ represents the probabilities of transitioning from $s_i$ to any other state (operator) seen among $N_Q$ and KNN n-grams.

While we choose the operator(s) with the highest probabilities in $M[s_i]$ to generate suggestions, the system assesses the certainty of these suggestions based on the probability distribution. This evaluation reflects the system's confidence that the suggested next operator is indeed the correct prediction.

In the least certain case, the probability distribution of transition from $s_i$ to any other state is uniform, making the system unsure about the next correct operator. To calculate the uncertainty score, we measure the deviation from the current probability distribution from the most uncertain distribution, i.e., the uniform distribution. This approach allows us to standardize the uncertainty among different transition matrices where the number of states differs.

We argue that increased variance in transition probabilities corresponds to reduced confidence in the model's ability to generate reliable suggestions. To calculate the variance among probability values in $M[s_i]$, we leverage the respective entropy:

$$H(M[s_i]) = -\sum_{0<j<|S|} M[s_i][j] \cdot \log_2(M[s_i][j]),$$

where $|S|$ is the number of all states, i.e., operators.

To obtain the maximum entropy, we calculate the entropy for a uniform distribution of probabilities:

$$H_{\max}(M[s_i]) = \log_2(|S|).$$

Ultimately, the uncertainty score is calculated as:

$$U(M[s_i]) = \frac{H(M[s_i])}{H_{\max}(M[s_i])}.$$

The uncertainty is between $[0,\ 1]$ and the maximum uncertainty, i.e., $U(M[s_i]) = 1$, is achieved when $H(M[s_i]) = H_{\max}(M[s_i])$.

## 7 Computational Complexity

We assess the computational complexity of FlowPilot by separating the system's operation into two phases: the offline phase and the online phase, which handles real-time suggestion requests.

**Offline Phase.** In the offline phase, the entire corpus of historical scientific workflows ($W$) is processed, which includes feature extraction, embedding generation, and SKB construction. Feature extraction primarily requires the generation of n-grams, as the remaining features can be fetched in $O(1)$ from their corresponding dictionary. The worst-case complexity of generating n-grams in a workflow DAG is $2^{|w|}$, where $|w|$ represents the number of workflow nodes. The time complexity of the embedding generation of `all-MiniLM-L6-v2` is $O(L^2d)$, with $L$ as the text length and $d$ as the dimension. Finally, the HNSW index generation takes $O(N \cdot logN)$ time, where $N$ is the number of elements in the index. Therefore, the total time complexity of the offline phase is $O(W \cdot 2^{|w|} \cdot (N \cdot logN + L^2d))$.

**Online Phase.** In the online phase, $N_q$ is the active path, which is given; therefore, the complexity depends on the feature extraction, KNN discovery, and Markov model training. The query complexity of SKB is $O(logN)$. To train the Markov model, we pair-wise evaluate the operators in the top-k candidate n-grams. Here, model training takes $O(n \cdot (n-1))$ time, with $n$ as the number of operators in the candidate n-grams. The suggestion requires finding the operator with maximum probability. As such, the online phase has a total time complexity of $O(L^2d) + O(n \cdot (n-1)) + O(n)$.

## 8 Experiments

Now, we evaluate FlowPilot by answering the following questions:

I) *what is the effectiveness of our suggestion generator?*
II) *how well does the system learn from user feedback?*
III) *how does the Markov chain modeling perform?*
IV) *how well does the system discover relevant domains?*

Below, we first outline our experimental setup and the baselines used in the comparative experiments. We then discuss the experimental results obtained in response to the above questions.

### 8.1 Experimental Setup

We conduct our experiments on various corpora of SWFs: **NF-Core Released** represents all NF-Core workflows that have a stable released version; **NF-Core UD** denotes those NF-Core workflows that are labeled as *under development* and do not have a stable version released yet; and **GitHub NF** contains all NextFlow repositories on GitHub[1]. We ignore the workflows with fewer than three operators to incorporate meaningful SWFs. Table 2 gives an overview of the corpora, including the number of all SWFs and the number of executable SWFs. We ensured that the GitHub NF corpus does not include the NF-Core workflows to avoid data leaks.

[1]GitHub corpus is obtained on 20.12.2024

Table 2. Corpora used in the experiments, including the name, number of SWFs involved, and number of executable SWFs.

| *Corpora* | *SWFs* | *Executable SWFs* |
|---|---|---|
| NF-Core Released | 72 | 55 |
| NF-Core UD | 38 | 25 |
| GitHub NF | 4982 | 873 |

Experiments are conducted by dividing data into two groups: training data, i.e., corpora that are used to train a model or generate the knowledge base, from which we generate suggestions; and query data, i.e., corpora from which we generate query n-grams. Note that training and query datasets are mutually exclusive. For instance, if the query workflow is NF-Core Released, the SKB and models are trained on NF-Core UD + GitHub NF.

Regarding query workflows, we choose workflows that have neither been queried nor indexed before. This maintains the trustworthiness of the results by avoiding data leakage. At each step, the index is updated based on the user feedback and the model's certainty (see Table 1). For the transfer learning experiment, the feedback loop is disabled. We initiate the experiment with the index structure already populated with the workflows from corpora other than the one used to generate the queries. For instance, if the query corpus is GitHub NF, the index is populated via all NF-C workflows.

To index n-grams from historical workflows, we extract all n-grams of various lengths. This enriches the index with various domain granularities, resulting in a flexible approach that can generate suggestions for queries with different degrees of completeness.

To generate ground truths from the query workflows, after extracting n-grams, we follow a masking strategy by considering the last element as the ground truth and trying to predict it based on the incomplete n-gram and the historical data stored in SKB. In our suggestion system, there is only a single ground truth operator. That is, at each point in time, there exists only one true next operator that the user implements. Therefore, traditional measures known from information retrieval scenarios with potentially multiple relevant elements in the result set, such as precision@k and recall@k, do not apply to our use case.

We further emphasize that semantic equality of the code is not a suitable basis for comparison in our use case. First, existing operators shall be reused as much as possible to foster the uptake of best practices, as otherwise, regardless of the semantic equality of implementations, the workflow will not pass the community vetting process. Second, while more than one tool can usually be used in different communities for various tasks, e.g., for short-read assembly, one can use Velvet or Minia, these tools typically differ in their results due to the adopted implementation strategies.

The code is available at our git repository.[2] We use *Python* for our implementations and run the experiments on a machine with 750GB main memory, 72 processing cores, and for LLM-based baselines, an NVIDIA A100 80GB PCIe GPU.

### 8.2 Baselines

To evaluate FlowPilot, we introduce the following baselines:

*8.2.1 End-to-End Baselines.* These are standalone systems.

**CodeT5+. [44]** The state-of-the-art code-generating model trained on code data for tasks including code completion [44]. We leverage the model with 16*b* parameters.

[2]https://github.com/hu-dbis/FlowPilot

**CodeLlama.** This is another code LLM, which is built on top of the Llama model. We use this model with $7b$ parameters.
**DeepSeek Coder.** It is code-generating model built on the DeepSeek language model. We leverage the model with $6.7b$ parameters.
**Llama4 Scout.** This model has a large context of size $10M$ tokens, making it ideal for prompting with large historical data.
**Bi-directional LSTM.** We train a bidirectional LSTM using the TensorFlow Keras API and evaluate the model over a total of 20 epochs. We built three layers: the embedding layer, the biLSTM layer, and a dense layer for prediction. The n-grams are randomly split into train and test sets with the 80% and 20% ratio, respectively.
**Frequent item set mining (FIM).** This baseline predicts the next operator through Frequent Item set Mining (FIM). It applies the Apriori algorithm to discover frequent operator sets. Using these operator sets, association rules are generated according to the confidence metric, with a pre-defined threshold of 0.3. The model then checks for matching antecedents and predicts the consequent operator accordingly. For FIM, we use the `MLxtend` implementation[3].
**Fine-Tuned Next Activity Prediction (FT-NAP) [34].** This approach modifies an LLM architecture by adding a classification layer to be able to predict, i.e., classify, the next operator. According to [34], the best results are obtained through the modification of Llama-3, so that we fine-tune this model for the baseline.
**Suggestion based on CCG [28].** CCG is a retrieval system for SWFs. It leverages a similarity metric that incorporates both, the workflow structure and data annotations. It transforms workflows into matrix representations, and measures the similarity of n-grams by computing distances between these matrices. To derive a suggestion for SWFs, we adopt the CCG similarity scoring when searching for KNN n-grams, and instantiate it with the Markov modeling and index updating as proposed by us in FlowPilot.
**FlowRecommender (FR) [4].** It leverages frequent sub-workflows and ranks them based on a similarity score based on the position of the sub-workflows in the workflow DAG.
**Auto-Suggest (AS) [49].** This baseline is inspired by the RNN model designed for seq-to-seq tasks. We use a more advanced GRU RNN model to train on the historical data and treat n-grams as sequences to be completed.
**SSKG [8].** Here, we incorporate stars of workflow repositories on GitHub to simulate workflow scores, and developers involved in each workflow repository to simulate social relations. Using our embedding and the retrieved social relations, we rank and obtain the KNN n-grams based on the $U_{ij}$ score.
**AutoML.** For this baseline, we model the suggestion problem as a multi-class classification. We build a training dataset of all sequences of operators and the next operator as the target. We train an autoML pipeline using AutoGluon [10].

*8.2.2 Baselines for ablation study.* These baselines are not end-to-end suggestion generators, but rather approaches to evaluate our Markov chain-based approach and domain discovery.
**Majority voting (MV).** This baseline uses the SKB to obtain the KNN n-grams for the query workflow and predicts the next operator based on majority voting, i.e., the most frequent operator.
**FIM-L.** The key difference between this baseline and FIM is that FIM-L searches for frequent operator sets within the KNN n-grams rather than the whole historical corpus. FIM-L aims at enhancing the precision by focusing only on contextually relevant patterns.

[3]https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/

Table 3. Accuracy comparison between FlowPilot and baselines. Experiments with - did not terminate within the time limit.

| *Corpus* | *FlowPilot* | *CodeLLM - Zero Shot* | | | *CodeLLM - Few Shot* | | | *FIM* |
|---|---|---|---|---|---|---|---|---|
| | | *CodeT5+* | *CodeLlama* | *DS Coder* | *CodeLlama* | *DS Coder* | *Llama4* | |
| NFC R. | **58%** | 01% | 02% | 01% | 04% | 09% | - | 03% |
| NFC UD | **58%** | 01% | 02% | 02% | 05% | 03% | - | 12% |
| GH NF | **56%** | 02% | 03% | 04% | - | - | - | 06% |

| *Corpus* | *FlowPilot* | *LSTM* | *FT-NAP [34]* | *CCG [28]* | *FR [4]* | *AS [49]* | *SSKG [8]* | *AutoML* |
|---|---|---|---|---|---|---|---|---|
| NFC R. | **58%** | 05% | 13% | 22% | 03% | 45% | 00% | 24% |
| NFC UD | **58%** | 09% | 21% | 38% | 01% | 53% | 04% | 18% |
| GH NF | **56%** | 08% | 23% | 07% | 02% | 38% | 00% | 29% |

**TF-IDF [17].** This approach generates n-gram embeddings based on the importance of each operator relative to its frequency in individual n-grams and across the entire corpus. This baseline is inspired by the SWORTS approach for workflow retrieval [17].

### 8.3 End-To-End Evaluation

Here, we compare FlowPilot to end-to-end systems. Code LLMs are prompted in two ways: zero- and few-shot learning. In the former, we provide the model with the incomplete implementation of the query workflow. We evaluate the completed code based on whether or not it contains the next operator. In the few-shot learning approach, in addition to the query workflow, we provide the flattened implementations of the complete historical data.

In this experiment, we measure accuracy, which represents the ratio of the queries with correct suggestions over all queries. It corresponds to recall@K, where there is only one relevant item. This is because recall@K is either 0, if the relevant item is not among the suggested ones, or 1, if it is. Averaging recall@k for all queries results in the accuracy measure.

Table 3 depicts the results of this experiment on three corpora. The - sign in the table indicates that the experiment did not finish within the 24-hour limit. The experiments show that FlowPilot outperforms all the baselines for all corpora by yielding the accuracy of 57% on average, indicating that 57% of the suggestions are correct. This is due to the effective SKB that allows the system to fetch the most relevant n-grams and generate the suggestion only based on those KNN elements. This reduces the noisy n-grams derived from irrelevant workflows that can negatively impact the prediction of the correct operator.

The second-best performing baseline is Auto-Suggest [49] with 45% accuracy on average. Our adaptation of this baseline uses a GRU RNN model to learn operator sequences. This approach still falls behind FlowPilot because it heavily relies on the data at runtime, which is unavailable at the development phase.

Although CCG uses a particular similarity measure, it still benefits from the Markov model and the index updates in FlowPilot. This is why it reaches a relatively high accuracy (22% on average) compared to other baselines. This experiment shows the benefit of our feature extraction component and KNN discovery method.

The autoML model that featurizes n-grams and learn a multi-class classifier to predict the next operator achieves 24% accuracy due to AutoGluon's effective search in finding the best pipeline. FR and SSKG, on the other hand, result in a very low accuracy. FR underperforms other baselines since it ignores the semantics of workflows and merely focuses on structure. On the other hand,

SSKG overfocuses on social relations and meta information in an interconnected community that ultimately the suggestions are similar to a random selection.

Code-LLMs, both zero- and few-shot, underperform FlowPilot by a large margin. The experiment shows that zero-shot Code-LLMs achieves a very low accuracy of 4% and less. This is even though these models have been trained on every code snippet on GitHub and more, highlighting the incompetence of current models in domains where publicly available data is insufficient.

Few-shot learning slightly increases the accuracy of Code-LLMs. This is because in few-shot learning baselines, the model has access to the corpus as an additional source of information. Doing so, the models leverage the context provided based on the historical data and generate better code completions. Note that Llama-4 does not terminate for any corpus in a 24-hour time limit. Yet, for a handful of finished cases, for which the model generated suggestions, we observe that the accuracy is as poor as with the other Code LLMs. This is also the case for the other few-shot learning experiments for GitHub NF, due to its larger number of SWFs.

However, fine-tuning an LLM for the specific task of operator suggestion, although it falls behind FlowPilot's performance, significantly improves the accuracy. FT-NAP achieves the accuracy of up to 23%. FIM and LSTM baselines generate better suggestions than non-tuned LLMs, but perform poorly compared to FlowPilot.

### 8.4 Suggestion Generator

In this experiment, we evaluate our suggestion generation model against several baselines. All of the baselines benefit from the online learning strategy proposed in Section 6.

Table 4 shows the precision of the approaches for various $K$ values, i.e., the number of retrieved nearest neighbors, and $R$, i.e., the number of suggestions generated per query. Precision is defined as the likelihood of suggesting the correct operator given that the operator is among the KNN.

The results indicate that FlowPilot with the average precision of 79% outperforms MV, FIM-L, Llama-3, Llama-4, and LSTM with 53%, 33%, 43%, 56%, and 12% precisions, respectively. Note that the relatively superior performance of the MV and Llama-4 is achieved through the effective SKB and the features we extracted in this paper to find the most similar n-grams.

The experiments also show that increasing $K$ does not necessarily increase the precision. Increasing the number of similar n-grams directly impacts the number of candidate operators to choose from. This increases the chance of making incorrect suggestions by introducing noise into the suggestion generation process. In fact, based on the average precisions, in the majority of the cases, $K = 10$ achieves the worst results. On the other hand, as expected, the precision increases with $R$ as the likelihood of the correct operator being among the suggested ones increases.

The newer Llama model, with more parameters, performs better than its predecessor, as expected. Despite this improvement, it still underperforms compared to FlowPilot, which is tailored for SWF suggestion tasks. This suggests that general-purpose LLMs are not optimal when applied directly to specialized tasks without adaptation. As shown in Table 3, LLMs achieve less than 5% accuracy, and even fine-tuning with the full corpus yields minimal gains. However, in a more controlled setting using carefully selected KNNs (Table 4), performance improves significantly.

The LSTM baseline yields the worst precision among the baselines. This is because the model requires a large number of examples to learn from. In this experiment, the LSTM's training set is limited to the K nearest neighbors, where $K \leq 10$, which is not enough for reliable prediction. The experiment also indicates that the LSTM yields lower precision for the GitHub corpus. This is because the operator set of NF-Core corpora does not highly overlap with that of NextFlow repositories on GitHub, and with a smaller training set, LSTM cannot predict the correct next operator.

Table 4. Precision comparison between FlowPilot and baselines. K indicates the number of nearest neighbor n-grams fetched from the SKB. R is the number of suggestions made by each baseline.

| *Corpus* | *K = 1* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | ***FlowPilot*** | | | ***Majority Voting*** | | | ***FIM-L*** | | |
| | **R=1** | **R=2** | **R=3** | **R=1** | **R=2** | **R=3** | **R=1** | **R=2** | **R=3** |
| NFC R. | **67%** | **76%** | **83%** | 61% | 67% | 71% | 19% | 28% | 25% |
| NFC UD | **62%** | **74%** | **82%** | 55% | 63% | 67% | 29% | 27% | 47% |
| GH NF | **72%** | **82%** | **88%** | 52% | 64% | 67% | 42% | 38% | 51% |
| | *K = 5* | | | | | | | | |
| | ***FlowPilot*** | | | ***Majority Voting*** | | | ***FIM-L*** | | |
| | **R=1** | **R=2** | **R=3** | **R=1** | **R=2** | **R=3** | **R=1** | **R=2** | **R=3** |
| NFC R. | **70%** | **85%** | **90%** | 45% | 53% | 56% | 24% | 41% | 31% |
| NFC UD | **65%** | **84%** | **90%** | 45% | 52% | 57% | 32% | 34% | 37% |
| GH NF | **72%** | **88%** | **93%** | 44% | 50% | 54% | 35% | 47% | 54% |
| | *K = 10* | | | | | | | | |
| | ***FlowPilot*** | | | ***Majority Voting*** | | | ***FIM-L*** | | |
| | **R=1** | **R=2** | **R=3** | **R=1** | **R=2** | **R=3** | **R=1** | **R=2** | **R=3** |
| NFC R. | **65%** | **83%** | **89%** | 42% | 47% | 51% | 18% | 26% | 31% |
| NFC UD | **64%** | **83%** | **89%** | 39% | 48% | 52% | 20% | 26% | 35% |
| GH NF | **70%** | **86%** | **91%** | 38% | 44% | 48% | 26% | 37% | 36% |

| *Corpus* | *K = 1* | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ***FlowPilot*** | | | ***Llama3*** | | | ***Llama4*** | | | ***LSTM*** | | |
| | **R=1** | **R=2** | **R=3** | **R=1** | **R=2** | **R=3** | **R=1** | **R=2** | **R=3** | **R=1** | **R=2** | **R=3** |
| NFC R. | **67%** | **76%** | **83%** | 28% | 41% | 43% | 30% | 62% | 70% | 13% | 24% | 33% |
| NFC UD | **62%** | **74%** | **82%** | 31% | 43% | 46% | 35% | 57% | 67% | 11% | 21% | 31% |
| GH NF | **72%** | **82%** | **88%** | 46% | 49% | 51% | 51% | 65% | 74% | 13% | 22% | 32% |
| | *K = 5* | | | | | | | | | | | |
| | ***FlowPilot*** | | | ***Llama3*** | | | ***Llama4*** | | | ***LSTM*** | | |
| | **R=1** | **R=2** | **R=3** | **R=1** | **R=2** | **R=3** | **R=1** | **R=2** | **R=3** | **R=1** | **R=2** | **R=3** |
| NFC R. | **70%** | **85%** | **90%** | 23% | 40% | 52% | 43% | 57% | 69% | 05% | 11% | 15% |
| NFC UD | **65%** | **84%** | **90%** | 29% | 47% | 58% | 36% | 59% | 69% | 06% | 11% | 14% |
| GH NF | **72%** | **88%** | **93%** | 40% | 55% | 65% | 44% | 62% | 74% | 04% | 11% | 14% |
| | *K = 10* | | | | | | | | | | | |
| | ***FlowPilot*** | | | ***Llama3*** | | | ***Llama4*** | | | ***LSTM*** | | |
| | **R=1** | **R=2** | **R=3** | **R=1** | **R=2** | **R=3** | **R=1** | **R=2** | **R=3** | **R=1** | **R=2** | **R=3** |
| NFC R. | **65%** | **83%** | **89%** | 22% | 36% | 46% | 42% | 53% | 63% | 03% | 05% | 07% |
| NFC UD | **64%** | **83%** | **89%** | 25% | 44% | 52% | 34% | 52% | 65% | 02% | 09% | 06% |
| GH NF | **70%** | **86%** | **91%** | 35% | 51% | 61% | 41% | 57% | 69% | 02% | 04% | 06% |

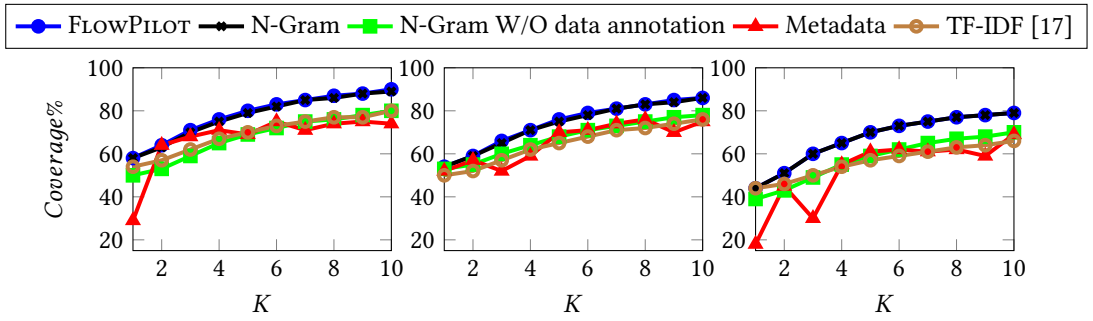



Fig. 3. Coverage experiment. NFC UD (left), NFC Released (middle), GitHub NF (right). K indicates the number of nearest neighbors retrieved from the SKB.

### 8.5 Domain Discovery Evaluation

To evaluate the effectiveness of the SKB and FlowPilot's ability in discovering relevant n-grams, we measure the ratio of the cases where the correct suggestion is in fact among the KNN fetched n-grams. We call this measure coverage. For this experiment, we compare index coverage for FlowPilot (with n-gram + metadata features) to baselines, including TF-IDF, and n-gram features (excluding metadata) with and without data annotations, and only metadata features.

Figure 3 depicts the results of this experiment. It shows that FlowPilot outperforms all baselines with the n-gram encoding performing very similarly with only 4% lower coverage on average. Among the remaining baselines, building the index using only workflow metadata performs as well as FlowPilot, and in some other cases, it leads to very low coverage. This shows that the workflow description can vary in quality, making the approach susceptible to noise. This quality issue is more obvious for the workflow in the GitHub NF corpus, because unlike NF-Core workflows, they do not go through an extensive quality assurance process.

Note that the overall coverage for the NF-Core UD experiment is higher than for the other two corpora. This is mainly because, in this experiment, we train on the NF-Core released corpus and suggest on NF-Core UD. We already know that the NF-Core released collection is well-curated and complete. Hence, training on the more complete NF-Core released and suggesting on NF-Core UD yields better coverage, as opposed to training on NF-Core UD and suggesting on NF-Core released. Also, the GitHub experiment results in lower coverage because NF-Core workflows are more homogeneous compared to the more heterogeneous GitHub corpus that drastically differs from the NF-Core workflows.

### 8.6 User Feedback Evaluation

We evaluate the impact of the feedback loop proposed in Section 6. To this end, we measure coverage, i.e., the ratio of the cases where the ground truth is among the KNN n-grams to evaluate the retrieval aspect of the system, and precision, i.e., ratio of the cases where the baseline can correctly predict the next operator. We compare FlowPilot against two baselines: transfer learning, in which, instead of incrementally updating the index with user feedback, we populate the index structure with the non-query corpora. Leave One Out Cross Validation (LOOCV): where we populate the index with all of the workflows in the same corpus except the query workflow. We repeat this process for every workflow and report the average. Table 5 shows the results of this experiment. It indicates that FlowPilot with the feedback loop results in 64% and 68% higher precision and 13% and 22% higher coverage compared to transfer learning and LOOCV baselines, respectively.

Table 5. Coverage and precision comparison between FlowPilot (with feedback loop), transfer learning (without feedback loop), and Leave-one-out-cross-validation (LOOCV) ($K = 10$).

| *Corpus* | *FlowPilot* | | *Transfer Learning* | | *LOOCV* | |
|---|---|---|---|---|---|---|
| | *Coverage* | *Prec.* | *Coverage* | *Prec.* | *Coverage* | *Prec.* |
| NFC R. | **86%** | **65%** | 27% | 54% | 16% | 44% |
| NFC UD | **90%** | **64%** | 21% | 54% | 11% | 41% |
| GH NF | **79%** | **70%** | 14% | 52% | 23% | 48% |

This shows that the feedback loop not only helps with introducing unforeseen operators (higher coverage) but also improves the effectiveness of correct suggestions (higher precision).

### 8.7 Runtime

As suggestion generation for workflow design is a real-time task to achieve an interactive experience, the runtime is of high importance. Thus, we evaluate FlowPilot's runtime compared to Llama-3 and Llama-4, as the runtime for the remaining baselines is comparable to FlowPilot. Figure 4 depicts the results of this experiment. The x-axis represents $K$, the number of similar n-grams, and the y-axis shows the runtime per iteration. In this experiment, the KNN retrieval and index update phases are identical. The only difference is that FlowPilot leverages the Markov-based prediction compared to the LLM-based prediction for Llama models.

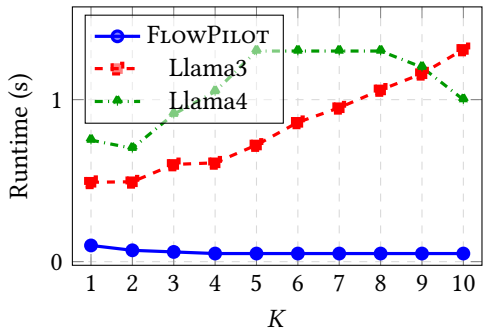



Fig. 4. Runtime comparison across $K$ on NFC R. corpus. $K$ is the number of nearest neighbors retrieved from SKB.

As the figure shows, FlowPilot achieves 13× and 19× lower runtime on average compared to Llama-3 and Llama-4, respectively. This is despite the fact that Llama models run on a high-end GPU, which the user might not have at their disposal. FlowPilot not only demonstrates higher effectiveness but also achieves better efficiency compared to the state-of-the-art LLMs, underscoring its suitability as a reliable suggestion system for developing SWFs.

### 8.8 System vs. User Operators

Each workflow engine, such as NextFlow, has a set of operators that is used to streamline the user-defined modules. These system operators are less versatile and are commonly used in SWFs, and therefore, their suggestions are expected to be more accurate. In this experiment, we measure the frequency and the accuracy of suggestions based on system and user operators.

Table 6. Freq. and precision for system and user operators.

| *Corpus* | *Frequency* | | *Precision* | |
|---|---|---|---|---|
| | *Sys. Op.* | *User Op.* | *Sys. Op.* | *User Op.* |
| NFC R. | 1,223 (8%) | 14,683 (92%) | 77% | 70% |
| NFC UD | 713 (14%) | 4,563 (86%) | 79% | 70% |
| GH NF | 4,249 (9%) | 43,448 (91%) | 82% | 71% |

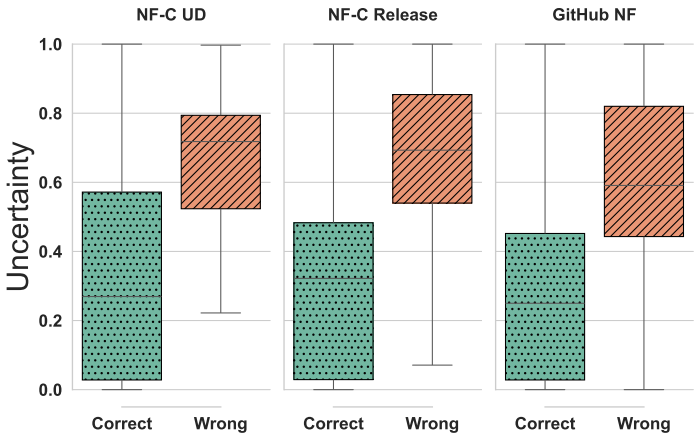



Fig. 5. The relation between uncertainty and correctness.

Table 6 shows that the frequency of system operators is a small fraction of user operators for NextFlow. Furthermore, the performance of our suggestion system is marginally better for system operators compared to user operators. This is expected as the variety of user operators is much higher than the system operators, therefore, the suggestion generator makes more mistakes in predicting more complex and yet less common user operators. Regardless, the results show that FlowPilot hardly overfits to the most frequent operators and is effective across different operator classes.

### 8.9 Uncertainty Measure

In this experiment, we investigate the relationship between the uncertainty of the Markov model and the generated suggestion. Figure 5 illustrates a box plot for uncertainties for both wrong and correct suggestions. As expected, the system generates more wrong predictions when the uncertainty is high, indicating that the KNN did not have enough information to generate a certain suggestion. On the other hand, the average uncertainty for the correct suggestions is lower for all three corpora.

## 9 Related Work

**SWF similarity and retrieval.** Existing solutions to assess the similarity of SWFs are typically based on the graph structure [38, 39], exploiting generic graph edit distances or recurrent patterns and motifs [14, 47]. While such techniques are useful, they neglect additional information, e.g., the

metadata of a workflow that is needed to derive accurate suggestions. As such, they are not directly suited for the use case addressed in our work.

Acknowledging these limitations, several approaches adopt semantically richer models. Ma et al. [28] proposed a similarity metric that, in addition to the workflow structure, incorporates data types and user-provided criteria per data type. If these criteria are not available, as it is generally the case, the similarity measure is a simple graph overlap based on exact data type matches. This is not practical as the data annotations, i.e., variable names, are arbitrary.

Other approaches focus purely on the data perspective and include the input and output operators of modules when calculating the similarity [18]. Yet, this way, the actual data flow of the workflow is neglected, so that the module context cannot be leveraged when using the measure to derive suggestions.

Compared to the above SWF measures, FlowPilot adopts a more comprehensive model for similarity scoring, which is based on embeddings that integrate information on the workflow structure, the data flow, and the workflow semantics.

SWORTS [17] has been proposed for discovering SWFs based on NL queries. It includes two phases: first, it retrieves candidate workflows employing textual similarity with a TF-IDF weighting scheme. Second, it leverages a deep learning approach to train a ranking model. Although SWORTS has not been designed for next-operator suggestions, using it for domain discovery, experimental results ( Figure 3), demonstrate that the embedding-based approach of FlowPilot yields consistently higher coverage.

Finally, similarity measures have also been proposed for business process models [4, 37]. Unlike SWFs, these models are mostly control-flow oriented and largely lack a detailed data-flow specification. As such, similarity assessment for business workflows is based on the graph-structure [2, 9, 30] or control-flow patterns [24, 41]. For instance, FlowRecommend [4] extracts all possible continuations of workflow graphs and scores them based on a syntactic similarity metric using the graph edit distance neglecting the semantic relation of workflows. Moreover, these measures are often augmented with similarity measures for operator labels [23, 25]. Yet, due to their complete abstraction from the data flow, these techniques are not suited for the domain of scientific workflows.

**Workflow suggestions.** While approaches for workflow retrieval, such as SWORTS [17], can be used for operator suggestion (matching operators of a query to existing workflows), they adopt a very limited notion of context for the suggestions.

Diao and Zhou [8] incorporated friendship relations among users to enhance workflow recommendations. In practice, workflow communities exhibit high interconnectivity, with users frequently participating in multiple projects. As such, the influence of social relations on recommendation performance is diminished.

Recently, it has been argued that LLMs are useful for a wide range of suggestion tasks, summarized as *suggestion as language processing* [15]. Thus, a plethora of LLM integrations in traditional recommender systems using collaborative filtering and content-based filtering has been proposed [26, 43, 54]. Yet, these techniques assume a large corpus of user interactions as a knowledge base, which is not available in the field of SWF design.

For business processes, next-activity prediction (NAP) is similar to our use case. LLMs, including Llama-3 and Mistral-2, have recently been used for NAP [34, 35]. The results indicate that LLMs generally struggle in NAP tasks, but outperform smaller encoder-based models when specifically trained for the task. FlowPilot takes a different approach, though, by complementing Code LLMs. As shown in our evaluation, see Table 3, the direct use of NAP techniques in our context does not yield satisfactory results.

Our work also links to retrieval augmented generation (RAG) systems that combine LLMs with a retrieval component to extract relevant content and improve the generation quality. While effective for general-purpose tasks like question answering or summarization, RAG systems are typically stateless and operate over unstructured text [27, 52]. In contrast, FlowPilot is designed for structured SWFs and iterative development within these workflows. As such, its ability to account for the suggestion context sets it apart from common RAG application scenarios.

Recently, automating data preparation and cleaning pipelines has received much attention. CtxPipe [13] relies on deep reinforcement learning to optimize downstream model performance in such pipelines, focusing on tabular data context. Comet [31] provides data-driven, step-by-step cleaning suggestions based solely on predicted impacts on ML accuracy, without considering workflow structure. Auto-Suggest [49] and Auto-Pipeline [51] predict next operators for tabular data from a set of limited pandas operators, e.g., Join, Pivot. In contrast, FlowPilot focuses on the SWF development phase, where the input data is not yet available. Operating as a code suggestion system, it proposes workflow operators based on structural and semantic information rather than data content. This makes our problem fundamentally different from that of data-driven systems, which can directly assess suggestion quality through performance of the pipeline in the downstream task.

**Seq-to-seq models.** These models aim at predicting the next token in a text sequence [42, 50]. FT-NAP [34] adapts a next token predicting model into a next operator predicting model by adding a classification layer. While next operator prediction resembles next token prediction, workflows differ fundamentally from text sequences. They are structured by data-flow dependencies, which may include commutativity, branching, and merging, rather than strictly linear control flow. Furthermore, standard seq-to-seq models, trained on large general-purpose corpora, are not well-suited for such low-data, domain-specific settings. To address this issue, Auto-Suggest [49] trains a separate model per operator as well as extracting features from tabular data to tailor the seq-to-seq model to pipeline recommendation tasks. This approach is not feasible for SWF suggestion, as the operator set is not known a priori, making it difficult to fix the size of the input sequence. Also, the input data is usually unavailable in the development phase of a workflow.

## 10 Conclusion

In this paper, we proposed FlowPilot, a suggestion system for the design of scientific workflows. Our work is motivated by the inability of state-of-the-art Code LLMs to generate meaningful completions of scientific workflows. To overcome their limitations, we complement the use of Code LLMs, prior to which, we identify the next operator that the user may intend to implement.

Specifically, FlowPilot relies on a similarity knowledge base (SKB) that indexes historical workflows to find the ones matching the current context. The SKB is based on a broad range of features that are extracted from common workflows. Based thereon, FlowPilot derives predictions using a Markov model and also incorporates user feedback on the suggested operators to update the SKB.

We evaluated FlowPilot with real-world collections of NextFlow workflows, against different baselines. Results indicate the effectiveness and efficiency of FlowPilot, which outperforms state-of-the-art Code-LLMs as well as other baselines that employ common data mining techniques and similarity measures for workflow retrieval.

## Acknowledgments

This work was funded by the German Research Foundation (DFG), CRC1404: “FONDA: Foundations of Workflows for Large-Scale Scientific Data Analysis”.

## References


[1] James Ashmore, bensouthgate, valentinoruggieri, Keerthana Bhaskaran, Asma Ali, nf-core bot, David Koppstein, Lathika Madhan Mohan, and Maxime U Garcia. 2024. nf-core/rnasplice: nf-core/rnasplice 1.0.4. doi:10.5281/ZENODO.15194198

[2] Catriel Beeri, Anat Eyal, Simon Kamenkovich, and Tova Milo. 2006. Querying Business Processes. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, Umeshwar Dayal, Kyu-Young Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang Kyun Cha, and Young-Kuk Kim (Eds.). ACM, 343–354. http://dl.acm.org/citation.cfm?id=1164158

[3] Sarah Cohen Boulakia and Ulf Leser. 2011. Search, adapt, and reuse: the future of scientific workflows. *SIGMOD Rec.* 40, 2 (2011), 6–16. doi:10.1145/2034863.2034865

[4] Bin Cao, Jianwei Yin, ShuiGuang Deng, Dongjing Wang, and Zhaohui Wu. 2012. Graph-based workflow recommendation: on improving business process modeling. In *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, Xue-wen Chen, Guy Lebanon, Haixun Wang, and Mohammed J. Zaki (Eds.). ACM, 1527–1531. doi:10.1145/2396761.2398466

[5] Susan B. Davidson and Juliana Freire. 2008. Provenance and scientific workflows: challenges and opportunities. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, Jason Tsong-Li Wang (Ed.). ACM, 1345–1350. doi:10.1145/1376616.1376772

[6] Ewa Deelman, Tom Peterka, Ilkay Altintas, Christopher D. Carothers, Kerstin Kleese van Dam, Kenneth Moreland, Manish Parashar, Lavanya Ramakrishnan, Michela Taufer, and Jeffrey S. Vetter. 2018. The future of scientific workflows. *Int. J. High Perform. Comput. Appl.* 32, 1 (2018), 159–175. doi:10.1177/1094342017704893

[7] Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. 2017. Nextflow enables reproducible computational workflows. *Nature biotechnology* 35, 4 (2017), 316–319.

[8] Jin Diao and Zhangbing Zhou. 2020. Scientific Workflow Recommendation Based on Service Knowledge Graph. In *2020 IEEE International Conference on Knowledge Graph, ICKG 2020, Online, August 9-11, 2020*, Enhong Chen and Grigoris Antoniou (Eds.). IEEE, 219–226. doi:10.1109/ICBK50248.2020.00040

[9] Remco M. Dijkman and Rik Eshuis. 2022. Process Querying Using Process Model Similarity. In *Process Querying Methods*, Artem Polyvyanyy (Ed.). Springer, 411–435. doi:10.1007/978-3-030-92875-9_14

[10] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander J. Smola. 2020. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. *CoRR* abs/2003.06505 (2020). arXiv:2003.06505 https://arxiv.org/abs/2003.06505

[11] Philip A Ewels, Alexander Peltzer, Sven Fillinger, Harshil Patel, Johannes Alneberg, Andreas Wilm, Maxime Ulysse Garcia, Paolo Di Tommaso, and Sven Nahnsen. 2020. The nf-core framework for community-curated bioinformatics pipelines. *Nature biotechnology* 38, 3 (2020), 276–278.

[12] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J. Miller. 2023. Semantics-aware Dataset Discovery from Data Lakes with Contextualized Column-based Representation Learning. *Proc. VLDB Endow.* 16, 7 (2023), 1726–1739. https://www.vldb.org/pvldb/vol16/p1726-fan.pdf

[13] 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. *Proc. ACM Manag. Data* 2, 6 (2024), 231:1–231:27. doi:10.1145/3698831

[14] Daniel Garijo, Pinar Alper, Khalid Belhajjame, Óscar Corcho, Yolanda Gil, and Carole A. Goble. 2014. Common motifs in scientific workflows: An empirical analysis. *Future Gener. Comput. Syst.* 36 (2014), 338–351. doi:10.1016/J.FUTURE.2013.09.018

[15] Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. 2022. Recommendation as Language Processing (RLP): A Unified Pretrain, Personalized Prompt & Predict Paradigm (P5). In *RecSys*, Jennifer Golbeck, F. Maxwell Harper, Vanessa Murdock, Michael D. Ekstrand, Bracha Shapira, Justin Basilico, Keld T. Lundgaard, and Even Oldridge (Eds.). ACM, 299–315. doi:10.1145/3523227.3546767

[16] GitHub. 2025. CoPilot. https://github.com/features/copilot.

[17] Yang Gu, Jian Cao, Shiyou Qian, and Wei Guan. 2023. SWORTS: A Scientific Workflow Retrieval Approach by Learning Textual and Structural Semantics. *IEEE Trans. Serv. Comput.* 16, 6 (2023), 4205–4219. doi:10.1109/TSC.2023.3315478

[18] Yang Gu, Jian Cao, Shiyou Qian, Nengjun Zhu, and Wei Guan. 2024. MANSOR: A module alignment method based on neighbor information for scientific workflow. *Concurr. Comput. Pract. Exp.* 36, 10 (2024). doi:10.1002/CPE.7736

[19] Shailja Gupta, Rajesh Ranjan, and Surya Narayan Singh. 2024. A Comprehensive Survey of Retrieval-Augmented Generation (RAG): Evolution, Current Landscape and Future Directions. *CoRR* abs/2410.12837 (2024). arXiv:2410.12837 doi:10.48550/ARXIV.2410.12837

[20] Huggingface. 2025. All MiniLM L6 v2. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2.

[21] Maliheh Izadi, Jonathan Katzy, Tim van Dam, Marc Otten, Razvan Mihai Popescu, and Arie van Deursen. 2024. Language Models for Code Completion: A Practical Evaluation. In *Proceedings of the 46th IEEE/ACM International Conference on*

*Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 79:1–79:13. doi:10.1145/3597503.3639138

[22] Sathvik Joel, Jie Wu, and Fatemeh Fard. 2024. A survey on llm-based code generation for low-resource and domain-specific programming languages. *ACM Transactions on Software Engineering and Methodology* (2024).

[23] Christopher Klinkmüller, Ingo Weber, Jan Mendling, Henrik Leopold, and André Ludwig. 2013. Increasing Recall of Process Model Matching by Improved Activity Label Matching. In *Business Process Management - 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 8094)*, Florian Daniel, Jianmin Wang, and Barbara Weber (Eds.). Springer, 211–218. doi:10.1007/978-3-642-40176-3_17

[24] Matthias Kunze, Matthias Weidlich, and Mathias Weske. 2011. Behavioral Similarity - A Proper Metric. In *Business Process Management - 9th International Conference, BPM 2011, Clermont-Ferrand, France, August 30 - September 2, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6896)*, Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf (Eds.). Springer, 166–181. doi:10.1007/978-3-642-23059-2_15

[25] Henrik Leopold, Mathias Niepert, Matthias Weidlich, Jan Mendling, Remco M. Dijkman, and Heiner Stuckenschmidt. 2012. Probabilistic Optimization of Semantic Process Model Matching. In *Business Process Management - 10th International Conference, BPM 2012, Tallinn, Estonia, September 3-6, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7481)*, Alistair Barros, Avigdor Gal, and Ekkart Kindler (Eds.). Springer, 319–334. doi:10.1007/978-3-642-32885-5_25

[26] Jianghao Lin, Xinyi Dai, Yunjia Xi, Weiwen Liu, Bo Chen, Hao Zhang, Yong Liu, Chuhan Wu, Xiangyang Li, Chenxu Zhu, Huifeng Guo, Yong Yu, Ruiming Tang, and Weinan Zhang. 2025. How Can Recommender Systems Benefit from Large Language Models: A Survey. *ACM Trans. Inf. Syst.* 43, 2 (2025), 28:1–28:47. doi:10.1145/3678004

[27] Zhongtan Lin, Xiaowen Zhang, Kaiqi Zhao, Xiaoling Lu, and Yuanyuan Zhang. 2025. STrajRAG: Supervised trajectory retrieval augmented generation for next POI recommendation with travel semantics. *Inf. Process. Manag.* 62, 6 (2025), 104235. doi:10.1016/J.IPM.2025.104235

[28] Yinglong Ma, Moyi Shi, and Jun Wei. 2015. Cost and accuracy aware scientific workflow retrieval based on distance measure. *Inf. Sci.* 314 (2015), 1–13. doi:10.1016/J.INS.2015.03.055

[29] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (2020), 824–836. doi:10.1109/TPAMI.2018.2889473

[30] Mirjam Minor, Alexander Tartakovski, and Ralph Bergmann. 2007. Representation and Structure-Based Similarity Assessment for Agile Workflows. In *ICCBR (Lecture Notes in Computer Science, Vol. 4626)*, Rosina Weber and Michael M. Richter (Eds.). Springer, 224–238. doi:10.1007/978-3-540-74141-1_16

[31] Sedir Mohammed, Felix Naumann, and Hazar Harmouch. 2025. Step-by-Step Data Cleaning Recommendations to Improve ML Prediction Accuracy. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, Alkis Simitsis, Bettina Kemme, Anna Queralt, Oscar Romero, and Petar Jovanovic (Eds.). OpenProceedings.org, 542–554. doi:10.48786/EDBT.2025.43

[32] Nhan Nguyen and Sarah Nadi. 2022. An Empirical Evaluation of GitHub Copilot's Code Suggestions. In *19th IEEE/ACM International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23-24, 2022*. ACM, 1–5. doi:10.1145/3524842.3528470

[33] Harshil Patel, Jonathan Manning, Phil Ewels, Maxime U Garcia, Alexander Peltzer, Rickard Hammarén, Olga Botvinnik, Adam Talbot, Friederike Hanssen, Gregor Sturm, nf-core bot, Matthias Zepper, Denis Moreno, Pranathi Vemuri, Mahesh Binzer-Panchal, Ezra Greenberg, silviamorins, Lorena Pantano, Robert Syme, Gavin Kelly, Lorenzo Sola, James A. Fellows Yates, Gabriel Lichtenstein, Jose Espinosa-Carrasco, rfenouil, Luke Zappia, Chris Cheshire, Edmund Miller, marchoeppner, and Peng Zhou. 2025. *nf-core/rnaseq: nf-core/rnaseq v3.21.0 - Mercury Macaw*. doi:10.5281/zenodo.17153746

[34] Adrian Rebmann, Fabian David Schmidt, Goran Glavas, and Han van der Aa. 2024. Evaluating the Ability of LLMs to Solve Semantics-Aware Process Mining Tasks. In *6th International Conference on Process Mining, ICPM 2024, Kgs. Lyngby, Denmark, October 14-18, 2024*. IEEE, 9–16. doi:10.1109/ICPM63005.2024.10680677

[35] Adrian Rebmann, Fabian David Schmidt, Goran Glavas, and Han van der Aa. 2025. On the Potential of Large Language Models to Solve Semantics-Aware Process Mining Tasks. *CoRR* abs/2504.21074 (2025). arXiv:2504.21074 doi:10.48550/ARXIV.2504.21074

[36] Mario Sänger, Ninon De Mecquenem, Katarzyna Ewa Lewinska, Vasilis Bountris, Fabian Lehmann, Ulf Leser, and Thomas Kosch. 2023. Large Language Models to the Rescue: Reducing the Complexity in Scientific Workflow Development Using ChatGPT. *CoRR* abs/2311.01825 (2023). arXiv:2311.01825 doi:10.48550/ARXIV.2311.01825

[37] Andreas Schoknecht, Tom Thaler, Peter Fettke, Andreas Oberweis, and Ralf Laue. 2017. Similarity of Business Process Models - A State-of-the-Art Analysis. *ACM Comput. Surv.* 50, 4 (2017), 52:1–52:33. doi:10.1145/3092694

[38] Johannes Starlinger, Sarah Cohen Boulakia, Sanjeev Khanna, Susan B. Davidson, and Ulf Leser. 2014. Layer Decomposition: An Effective Structure-Based Approach for Scientific Workflow Similarity. In *10th IEEE International Conference on e-Science, eScience 2014, Sao Paulo, Brazil, October 20-24, 2014*. IEEE Computer Society, 169–176. doi:10.1109/ESCIENCE.2014.19

[39] Johannes Starlinger, Bryan Brancotte, Sarah Cohen Boulakia, and Ulf Leser. 2014. Similarity Search for Scientific Workflows. *Proc. VLDB Endow.* 7, 12 (2014), 1143–1154. doi:10.14778/2732977.2732988

[40] Mario Sänger, Ninon De Mecquenem, Katarzyna Ewa Lewińska, Vasilis Bountris, Fabian Lehmann, Ulf Leser, and Thomas Kosch. 2024. A qualitative assessment of using ChatGPT as large language model for scientific workflow development. *GigaScience* 13 (06 2024), giae030. arXiv:https://academic.oup.com/gigascience/article-pdf/doi/10.1093/gigascience/giae030/60704863/giae030.pdf doi:10.1093/gigascience/giae030

[41] Boudewijn F. van Dongen, Remco M. Dijkman, and Jan Mendling. 2008. Measuring Similarity between Business Process Models. In *Advanced Information Systems Engineering, 20th International Conference, CAiSE 2008, Montpellier, France, June 16-20, 2008, Proceedings (Lecture Notes in Computer Science, Vol. 5074)*, Zohra Bellahsene and Michel Léonard (Eds.). Springer, 450–464. doi:10.1007/978-3-540-69534-9_34

[42] Jia Wang, Tong Sun, Benyuan Liu, Yu Cao, and Hongwei Zhu. 2019. CLVSA: A Convolutional LSTM Based Variational Sequence-to-Sequence Model with Attention for Predicting Trends of Financial Markets. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, Sarit Kraus (Ed.). ijcai.org, 3705–3711. doi:10.24963/IJCAI.2019/514

[43] Xin Wang, Hong Chen, Zirui Pan, Yuwei Zhou, Chaoyu Guan, Lifeng Sun, and Wenwu Zhu. 2025. Automated Disentangled Sequential Recommendation with Large Language Models. *ACM Trans. Inf. Syst.* 43, 2 (2025), 29:1–29:29. doi:10.1145/3675164

[44] Yue Wang, Hung Le, Akhilesh Gotmare, Nghi D. Q. Bui, Junnan Li, and Steven C. H. Hoi. 2023. CodeT5+: Open Code Large Language Models for Code Understanding and Generation. In *EMNLP*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 1069–1088. doi:10.18653/V1/2023.EMNLP-MAIN.68

[45] Zuozhi Wang, Yicong Huang, Shengquan Ni, Avinash Kumar, Sadeem Alsudais, Xiaozhen Liu, Xinyuan Lin, Yunyan Ding, and Chen Li. 2024. Texera: A System for Collaborative and Interactive Data Analytics Using Workflows. *Proc. VLDB Endow.* 17, 11 (2024), 3580–3588. doi:10.14778/3681954.3682022

[46] David Luis Wiegandt, Johannes Starlinger, and Ulf Leser. 2016. Graph n-grams for Scientific Workflow Similarity Search. In *Proceedings of the Conference "Lernen, Wissen, Daten, Analysen" (CEUR Workshop Proceedings, Vol. 1670)*, Ralf Krestel, Davide Mottin, and Emmanuel Müller (Eds.). CEUR-WS.org, 213–224. https://ceur-ws.org/Vol-1670/paper-50.pdf

[47] CR Wijesinghe and AR Weerasinghe. 2020. Mining frequent patterns in bioinformatics workflows. *Int. J. Bioscience, Biochemistry, Bioinformatics* 10, 4 (2020), 161–169.

[48] Qinyun Wu, Chao Peng, Pengfei Gao, Ruida Hu, Haoyu Gan, Bo Jiang, Jinhe Tang, Zhiwen Deng, Zhanming Guan, Cuiyun Gao, Xia Liu, and Ping Yang. 2024. RepoMasterEval: Evaluating Code Completion via Real-World Repositories. *CoRR* abs/2408.03519 (2024). arXiv:2408.03519 doi:10.48550/ARXIV.2408.03519

[49] Cong Yan and Yeye He. 2020. Auto-Suggest: Learning-to-Recommend Data Preparation Steps Using Data Science Notebooks. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 1539–1554. doi:10.1145/3318464.3389738

[50] Weichao Yan, Hao Ma, and Zaiyue Yang. 2025. Segmented Sequence Prediction Using Variable-Order Markov Model Ensemble. *IEEE Trans. Knowl. Data Eng.* 37, 3 (2025), 1425–1438. doi:10.1109/TKDE.2024.3522975

[51] Junwen Yang, Yeye He, and Surajit Chaudhuri. 2021. Auto-Pipeline: Synthesize Data Pipelines By-Target Using Reinforcement Learning and Search. *Proc. VLDB Endow.* 14, 11 (2021), 2563–2575. doi:10.14778/3476249.3476303

[52] Yahe Yang and Chengyue Huang. 2025. Tree-based RAG-Agent Recommendation System: A Case Study in Medical Test Data. *CoRR* abs/2501.02727 (2025). arXiv:2501.02727 doi:10.48550/ARXIV.2501.02727

[53] Burak Yetistiren, Isik Ozsoy, and Eray Tuzun. 2022. Assessing the quality of GitHub copilot's code generation. In *PROMISE*, Shane McIntosh, Weiyi Shang, and Gema Rodríguez-Pérez (Eds.). ACM, 62–71. doi:10.1145/3558489.3559072

[54] Zihuai Zhao, Wenqi Fan, Jiatong Li, Yunqing Liu, Xiaowei Mei, Yiqi Wang, Zhen Wen, Fei Wang, Xiangyu Zhao, Jiliang Tang, and Qing Li. 2024. Recommender Systems in the Era of Large Language Models (LLMs). *IEEE Trans. Knowl. Data Eng.* 36, 11 (2024), 6889–6907. doi:10.1109/TKDE.2024.3392335

Received July 2025; revised October 2025; accepted November 2025