STARFLOW: GENERATING STRUCTURED WORKFLOW OUTPUTS FROM SKETCH IMAGES

Patrice Bechard[1] **Chao Wang**[1] **Amirhossein Abaskohi**[1,2]
Juan Rodriguez[1,3,4] **Christopher Pal**[1,3,5,6] **David Vazquez**[1]
Spandana Gella[1*] **Sai Rajeswar**[1,3*] **Perouz Taslakian**[1*]
[1]ServiceNow [2]University of British Columbia [3]Mila
[4]École de Technologie Supérieure [5]CIFAR AI Chair [6]Polytechnique Montréal

ABSTRACT

Workflows are a fundamental component of automation in enterprise platforms, enabling the orchestration of tasks, data processing, and system integrations. Despite being widely used, building workflows can be complex, often requiring manual configuration through low-code platforms or visual programming tools. To simplify this process, we explore the use of generative foundation models, particularly vision-language models (VLMs), to automatically generate structured workflows from visual inputs. Translating hand-drawn sketches or computer-generated diagrams into executable workflows is challenging due to the ambiguity of free-form drawings, variations in diagram styles, and the difficulty of inferring execution logic from visual elements. To address this, we introduce STARFLOW, a framework for generating structured workflow outputs from sketches using vision-language models. We curate a diverse dataset of workflow diagrams—including synthetic, manually annotated, and real-world samples to enable robust training and evaluation. We finetune and benchmark multiple vision-language models, conducting a series of ablation studies to analyze the strengths and limitations of our approach. Our results show that finetuning significantly enhances structured workflow generation, outperforming large vision-language models on this task.

1 INTRODUCTION

Workflows play a crucial role in automating business processes, orchestrating data flows, and integrating enterprise applications. They enable organizations to streamline operations, reduce manual effort, and enforce business logic across complex systems (MuleSoft, a Salesforce Company, 2025; ServiceNow, 2025; Microsoft, 2025). Despite their ubiquity, workflow creation remains a challenging task, requiring manual configuration of processes through low-code platforms or visual programming environments. While these tools offer greater accessibility than traditional programming, they still demand a deep understanding of system logic, data dependencies, and execution rules.

An intuitive alternative would be the ability to generate structured workflows directly from visual representations, such as hand-drawn sketches or diagrams, as portrayed in Figure 1. However, this problem is inherently difficult due to the ambiguity of sketches, variations in diagramming conventions, and the complexity of extracting structured execution logic from visual elements.

In this work, we introduce **STARFLOW**, a framework designed to generate structured workflow representations from sketch-based inputs using vision-language models (VLMs). Our approach involves curating a diverse dataset comprising synthetic, manually annotated, and real-world workflow diagrams, which we use to finetune multiple vision-language models. To evaluate the performance of our approach, we use a *flow similarity* metric that measures the structural fidelity of generated workflows based on the tree representation of the workflow and tree edit distance. Our results demonstrate that finetuning significantly enhances the ability of VLMs to generate structured workflows, outperforming general-purpose models on this specialized task.

[*]Equal supervision.

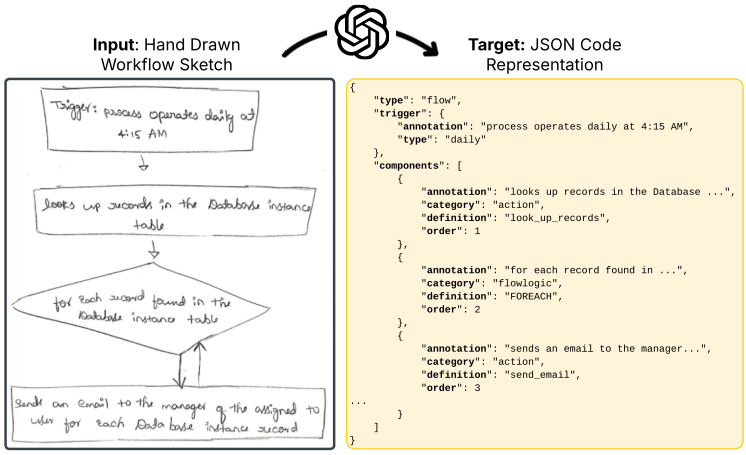


Figure 1: **The task of sketch to workflow.** Given an input image representing a business process, the task is to convert the logic of the diagram found in the image into a structured JSON output describing the execution logic of the workflow, including the appropriate trigger, actions, and inputs.

By addressing the limitations of existing workflow creation methods and demonstrating the effectiveness of vision-language models in this domain, our work represents a step toward making workflow automation more intuitive and accessible. Our key contributions are as follows:

- We introduce StarFlow, a framework for converting hand-drawn and digital workflow sketches into structured representations, enabling seamless workflow automation.
- We build a diverse dataset of workflow diagrams, spanning synthetic, human-annotated, and real-world samples, to enhance training and evaluation.

To foster future research and reproducibility, we will **open-source all resources**, including models and their implementations, code used for training, datasets, and evaluation metrics.

2 Related Work

2.1 Structured Output and Code Generation

Language models trained on code (e.g. Chen et al., 2021; Li et al., 2023; Roziere et al., 2023; Hui et al., 2024; Zhu et al., 2024) have seen significant advancements in recent years, improving various aspects of software development, including code generation (Nijkamp et al., 2022; Jiang et al., 2024; Rodriguez et al., 2024b), comprehension (Feng et al., 2020; Lu et al., 2021), and translation tasks (Lachaux et al., 2020; Yan et al., 2023). These models utilize large-scale source code datasets (e.g. Kocetkov et al., 2022) to learn programming syntax and semantics, enabling them to produce functional and syntactically correct code snippets from natural language prompts.

Evaluation of code generation models is notoriously difficult. Metrics such as the HumanEval benchmark (Chen et al., 2021) aim to evaluate a model's ability to generate functionally correct code solutions. Other metrics, such as CodeBLEU (Ren et al., 2020), extend the traditional BLEU score (Papineni et al., 2002) by incorporating code-specific features such as syntax and data flow, offering a more nuanced evaluation of code generation quality. In this work, we draw inspiration from CodeBLEU and introduce a flow similarity metric based on tree representation and tree edit distance (Zhang & Shasha, 1989).

2.2 Multimodal Large Language Models

Vision-language models (VLMs) (e.g. Alayrac et al., 2022; Liu et al., 2023; Agrawal et al., 2024; Wang et al., 2024; Dubey et al., 2024) have made significant strides in integrating visual and textual data, enabling more sophisticated multimodal understanding. These models excel at various tasks, including image captioning (Lin et al., 2014; Vinyals et al., 2015), visual question answering (Antol et al., 2015; Hudson & Manning, 2019; Yue et al., 2024), and document understanding (Rodriguez et al., 2024a; Tong et al., 2025).

One task that remains challenging for VLMs is to generate code or structured outputs based on a screenshot or diagram (Liu et al., 2022; Shukla et al., 2023; Shi et al., 2025; Rodriguez et al., 2024a; Herrera-Camara & Hammond, 2017). For example, Shi et al. (2025) introduce a benchmark to assess the performance of VLMs on generating code to reproduce charts. Closely related to our work, Liu et al. (2022) propose a two-step process to generate code from a flowchart via two distinct models, one to extract the structure of the diagram, and the other to generate executable code from pseudocode. In this paper, we focus on generating structured workflows in JSON format from hand-drawn or computer generated sketches.

2.3 Workflow Generation

Recent work on workflow generation from textual inputs has demonstrated significant advancements in the field of automated task planning and execution. Approaches relying on retrieval-augmented generation and task decomposition have been shown to be effective for solving this problem (Béchard & Ayala, 2024; Bassamzadeh & Methani, 2024; Ayala & Béchard, 2024). Other notable efforts include Zeng et al. (2023), who built models to generate workflows for specific applications, Fan et al. (2024) who develop a synthetic data pipeline used to train a workflow generator, and Cai et al. (2023) who built a graphical user interface allowing a user to build and edit a workflow with the assistance of an LLM. In this work, we focus on generating workflows from hand-drawn sketches and computer generated diagrams instead of doing so from textual instructions.

3 Methodology

In this section, we go over the dataset creation process and how we evaluate generated flows. We first present a quick overview of what workflows are. We then discuss how we build synthetic workflows by finding patterns frequently found in ones that appear in the real-world. Finally, we highlight how we programmatically create diagrams for these workflows, and how we use these samples as a basis for the human-annotated data.

3.1 The Anatomy of a Workflow

Workflows are automated processes that consist of a sequence of reusable *actions* that perform operations on a user's data. Within a workflow, actions are intertwined with *flow logic* elements, such as conditions and loops, that control the execution of the workflow. A workflow typically includes a *trigger* that determines when the execution starts. Alternatively, a *subflow* consists of the same actions and flow logic as a workflow, but does not include a trigger. Subflows are meant to be called by workflows or other subflows, similar to how functions are used in programming languages.

Workflows can be triggered in a variety of ways. For example, a workflow can start after a certain interval of time has passed, when a record has been updated in a given table, or when an email is received, to name a few. The actions found in workflows can also perform a variety of operations on behalf of a user. For example, they can look up a set of records in a given table, make updates to records, send emails, connect to third-party APIs, and much more.

3.2 Synthetic Workflow Generation

Real world workflows are often built using a distinct set of design patterns. To build our synthetic workflow generation pipeline, we implemented a heuristic that can build workflows using a set of flow logic elements (e.g. IF, ELSE, FOREACH) along with actions and subflows sampled either deterministically or stochastically based on the pattern. Algorithm 1 presents a simplified look at the code used for creating a workflow following the *Scheduled Loop* pattern, which performs actions on multiple records at predefined time intervals.

Algorithm 1: Pseudocode for Scheduled Loop pattern

```
Select a random Scheduled trigger
Add a Look Up Records action for a table
Add a FOREACH flow logic
if random() < P_IF then
    Add an IF flow logic
end if
Select a random action related to the table
if IF statement exists and random() < P_IF then
    Add an ELSE flow logic
    Select another related action
end if
```

After creating the workflows, we generate natural language annotations for each step using a large language model — in our case, we used Llama 3.1 70B Instruct (Dubey et al., 2024). We represent the resulting workflows in JSON format, which serves as the generation target for the VLM. Figure 6 in Appendix A presents an example flow generated using the *Scheduled Loop* heuristic.

Once the synthetic workflows are generated, we proceed to creating variants of these samples using a variety of methods, thus obtaining workflow diagrams of five different flavors: SYNTHETIC, MANUAL, DIGITAL, WHITEBOARD, and USER INTERFACE. We describe the generation process of each in the next section.

3.3 CREATING WORKFLOW DIAGRAMS

SYNTHETIC workflows are created by programmatically generating a graph representation of each workflow using Graphviz (Ellson et al., 2002), including random variations in graph orientation and edge style. For example, the graph representation of the workflow in Figure 6 is shown in Figure 10a (Appendix K).

To create the USER INTERFACE workflows, we further render the programmatically generated flows using ServiceNow's native visualization tool, as illustrated in Figure 10e (Appendix K). This offers an alternative representation of the flows within an environment that closely aligns with potential deployment scenarios.

Table 1: Dataset distribution across splits. Samples are collected from a variety of sources, ranging from visualizations generated synthetically to hand-drawn samples. Examples for each type of sample can be found in Appendix K.

Source	Train	Valid	Test
SYNTHETIC	12,376	1,000	1,000
MANUAL	3,035	333	865
DIGITAL	2,613	241	701
WHITEBOARD	484	40	46
USER INTERFACE	373	116	87
Total	18,881	1,730	2,699

The three workflow types MANUAL, DIGITAL, and WHITEBOARD are created by human annotators. We contracted an external vendor to recruit annotators to create flow diagrams based on the synthetically generated ones. The annotators were given these graph representations and were asked to create flow diagrams for each graph sample using either digital tools (DIGITAL), or by drawing the graph on paper (MANUAL) or on a whiteboard or blackboard (WHITEBOARD). Details regarding the human annotators can be found in Appendix D. Figure 10c in Appendix K presents such an example for the same flow found in Figure 6.

For each flow JSON in our dataset, we generate one or more images using the approach described above. We then divide the samples according to the flow JSON, ensuring that no flows are shared between the different dataset splits. The number of samples generated for each sample type can be found in Table 1.

4 EXPERIMENTS

In this section, we conduct experiments to assess the capabilities of various open-weight and proprietary VLM models on the Sketch-to-Workflow task and its evaluation metrics. Additionally, we examine whether finetuning improves performance on the downstream task.

4.1 MODELS

We perform our experiments using a variety of frontier models as well as open-weight alternatives. We evaluate the following proprietary models: GPT-4o and GPT-4o-mini (Hurst et al., 2024), Claude-3.7-Sonnet (Anthropic, 2024), Gemini-2.0-Flash (Team et al., 2023). We put these models head-to-head against a set of strong open-weights alternatives, namely Pixtral (Agrawal et al., 2024), LLaMA 3.2 Vision (11B and 90B) (Dubey et al., 2024), Phi-3.5 (Abdin et al., 2024), and Qwen2.5-VL (3B, 7B and 72B) (Bai et al., 2025). Additionally, we finetune the smaller variants of the open-weight models and observe the resulting improvements on downstream tasks. Training details for the finetuned models can be found in Appendix E.

Table 2: **Flow quality metrics comparison across different models**. We compare proprietary models against open-weight models and their finetuned versions, with higher values indicating better performance for each metric. The best metric within each model category is highlighted in **bold**, and the runner up is underlined if there are more than two models in that category. Models are categorized by size: blue for models smaller than 4B parameters, orange for models between 4B and 12B parameters, green for models larger than 12B parameters, and gray for proprietary models. We evaluate models by making a single workflow generation call to the language model.

Model	FlowSim w/ inputs	FlowSim no inputs	TreeBLEU w/ inputs	TreeBLEU no inputs	Trigger match	Component match
Open-weights Models						
Qwen-2.5-VL-3B-Instruct	**0.410**	**0.384**	**0.360**	**0.329**	0.027	**0.201**
Phi-3.5-Vision-4B-Instruct	0.364	0.346	0.337	0.295	**0.079**	0.193
Phi-4-Multimodal-6B-Instruct	0.465	0.404	0.394	0.298	0.054	0.244
Qwen-2.5-VL-7B-Instruct	0.614	0.538	0.562	0.508	0.036	**0.280**
LLaMA-3.2-11B-Vision- Instruct	0.466	0.435	0.416	0.382	0.075	0.239
Pixtral-12B	**0.632**	**0.582**	**0.617**	**0.541**	**0.088**	0.261
Qwen-2.5-VL-72B-Instruct	**0.710**	**0.643**	**0.703**	**0.655**	0.325	**0.305**
LLaMA-3.2-90B-Vision-Instruct	0.687	0.603	0.681	0.627	**0.328**	0.286
Proprietary Models						
GPT-4o-Mini	0.642	0.617	0.650	0.623	0.254	0.305
GPT-4o	**0.786**	0.707	0.794	0.718	0.282	0.317
Claude-3.7-Sonnet	0.763	0.679	0.769	0.701	0.318	0.305
Gemini Flash 2.0	0.780	**0.713**	**0.798**	**0.743**	**0.466**	**0.329**
Finetuned Models						
Qwen-2.5-VL-3B-Instruct	**0.941**	**0.911**	**0.941**	**0.902**	**0.775**	**0.909**
Phi-3.5-Vision-4B-Instruct	0.917	0.882	0.917	0.869	0.703	0.874
Phi-4-Multimodal-6B-Instruct	0.939	0.908	0.940	0.901	0.770	0.907
Qwen-2.5-VL-7B-Instruct	**0.957**	**0.927**	**0.956**	**0.920**	**0.819**	**0.934**
LLaMA-3.2-11B-Vision-Instruct	0.955	0.924	0.954	0.915	0.805	**0.934**
Pixtral-12B	0.952	0.919	0.950	0.908	0.753	0.930

4.2 Evaluation of Generated Workflows

Assessing the quality of generated flows presents challenges similar to those in evaluating generated code. In this work, we report four types of metrics that provide a comprehensive evaluation by capturing different aspects of flow generation. The metrics we report are Flow Similarity ($FlowSim$), Tree BLEU ($TreeBLEU$), Trigger Match (TM), and Component Match (CM). For *Flow Similarity*, we follow the methodology used in Ayala & Béchard (2024): we decompose generated workflows into trees and compute the tree edit distance using the algorithm from Zhang & Shasha (1989). We normalize the obtained tree edit distance by the number of nodes in each tree to obtain a score between 0 and 1.

$$\text{FlowSim}(F, F_r) = 1 - \frac{TED(F, F_r)}{|F| + |F_r|} \tag{1}$$

where F, F_r denote the given flow and the reference flow, respectively.

We use a custom weighting scheme that assigns greater weight to changes affecting actions than those affecting inputs. Figure 9 in Appendix J illustrates the tree decomposition derived from the flow JSON defined in Figure 6.

We also use a variant of *TreeBLEU* (Gui et al., 2025) that leverages our tree decomposition to assess structural hierarchy recall between flows.

$$\text{TreeBLEU}(F, F_r) = \frac{|S(F) \cap S(F_r)|}{|S(F)|} \tag{2}$$

where $S(.)$ denotes the set of 1-height subtrees.

To ensure fairness, we exclude subtrees of height 1 that are always present — specifically, the `Flow` $\rightarrow$ `Trigger` and `Flow` $\rightarrow$ `Components` edges — so that empty flows without triggers or components receive a score of zero.

Trigger Match measures the percentage of cases where the model correctly predicts the trigger from the sample. *Component Match*, on the other hand, computes the intersection between the predicted

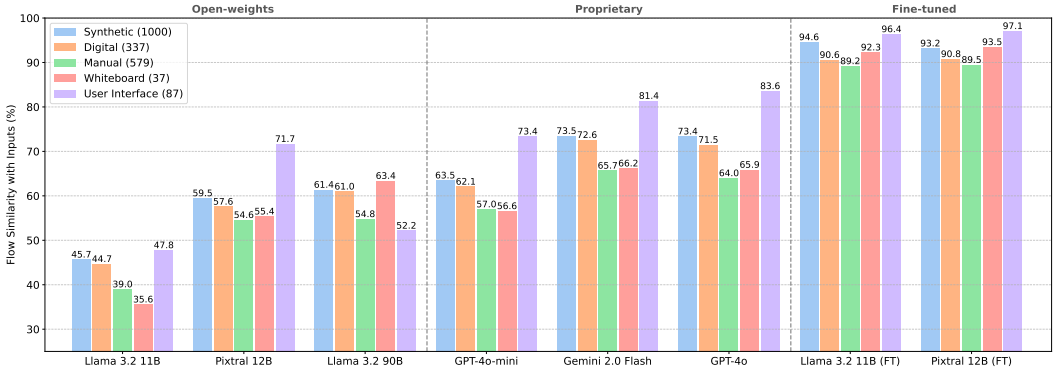


Figure 2: **Performance of each model per type of sample**. We report the FlowSim with inputs results. Number of supporting examples for each sample type is shown in parenthesis. Examples for each type of sample can be found in Appendix K.

and target components, normalized by their union. This metric evaluates the model's ability to predict the correct components in an order-agnostic manner, akin to the *bag-of-components* metric from (Béchard & Ayala, 2024). Equation 3 depicts both the Trigger Match and Component Match metrics.

$$\text{TM} = \mathbf{1}_{\{T_F = T_{F_r}\}} \qquad \text{CM} = \frac{|C_F \cap C_{F_r}|}{|C_F \cup C_{F_r}|} \tag{3}$$

where T_F and T_{F_r} denote the trigger of the given and reference flows, and C_F and C_{F_R} denote the set of components in each flow.

4.3 Sketch to Workflow

In this section, we assess the performance of models listed in Section 4.1 on the task of sketch to workflow generation. We evaluate models that are proprietary and ones that have open weights, across a variety of model sizes. Our experiments indicate that (1) most proprietary models perform better than open-weights ones without any domain-specific training, and that (2) finetuning on StarFlow helps open-weights models outperform proprietary models. Our results are summarized in Table 2.

Our results show that most proprietary models perform well on the workflow generation task. As expected, GPT-4o-mini underperforms compared to larger models, likely due to its smaller size. Among open-weight models, all models from the Qwen2.5-VL family of models perform remarkably well against models of similar sizes. Pixtral is another strong model for its size, nearly matching the performance of the larger Llama variant. In addition, performance trends remain consistent across different evaluation metrics. Across all models, scores for FlowSim and TreeBLEU are closely aligned, whether or not input conditions are considered. Additionally, finetuned models perform strongly on the trigger match TM metric, whereas proprietary and non-finetuned open-weight models lag further behind.

Finetuning significantly improves performance, surpassing all baselines by a substantial margin. In particular, the finetuned version of Qwen-2.5-VL-7B achieves notably high scores compared to all other models, closely followed by Llama 3.2 11B and Pixtral-12B. We hypothesize that finetuned models acquire crucial domain knowledge during training, which proprietary models struggle to replicate without additional external information. For example, when prompted with an image representing a flow for creating a user in Microsoft Azure Active Directory, a proprietary model must infer the type, definition, and scope of the relevant component. If the model predicts a component of type `action` with definition name `create_user` in scope `ms_azure_active_directory`, but the actual answer is a component of type `action` with definition name `create_a_user` in scope `sn_ms_ad_spoke`, it receives a score of zero.

Finetuned models benefit from exposure to such components during training, allowing them to memorize proper naming conventions of different components and improve accuracy. There are several potential ways to mitigate this issue. One approach is to integrate tool calls, enabling the VLM to retrieve relevant components during generation. Another is to incorporate retrieval-augmented generation (RAG) by extracting relevant details directly from images. Alternatively, breaking down the

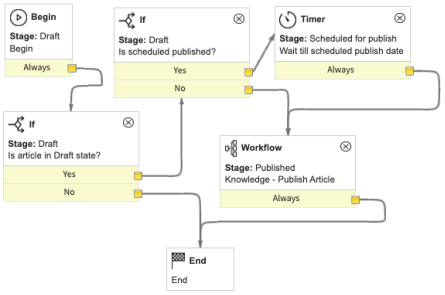


Figure 3: **Screenshot of a workflow**. The screenshot was taken from a visualization platform for which we do not have examples in the training dataset.

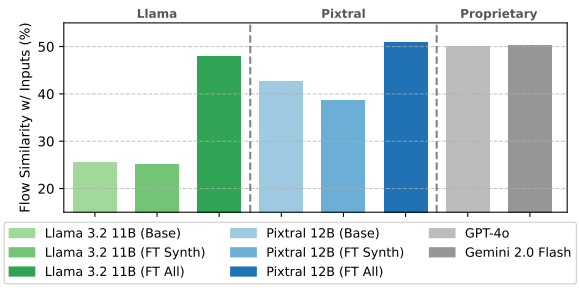


Figure 4: **Out-of-distribution generalization results.** Models finetuned on diverse data perform on par with proprietary models.

task into smaller subtasks could facilitate more effective retrieval of contextual information, helping to ground VLMs during generation.

4.4 Evaluation by Sample Subpopulation

In section 4, we observed that models finetuned with StarFlow generally outperform ones that are not finetuned in workflow generation. One question is whether these models perform better across all types of samples. To answer this question, we evaluate a subset of the models discussed in Section 4.3 on a stratified version of our dataset. Results are shown in Figure 2.

We find that all models experience a drop in performance on the Manual samples compared to other types of samples, closely followed by Whiteboard samples. Intuitively, these images are harder to interpret as they require the model to read handwritten text in order to properly select components used to generate the workflow. On the other hand, we find that User Interface screenshots and Synthetic samples are the easiest samples. Since these samples are rendered automatically, we hypothesize that they contain the least amount of ambiguity regarding the execution logic of the workflow. User Interface samples contain more textual information than other types, as the interface interprets the flow and presents additional details about the triggers and components (see Figure 10e in Appendix K). This extra context can make the task easier for models.

We include additional ablations in Appendix B, including a study of the impact of the orientation of the sample as well as the size of the input image on the performance of the model.

4.5 Generalization Beyond Training Distribution

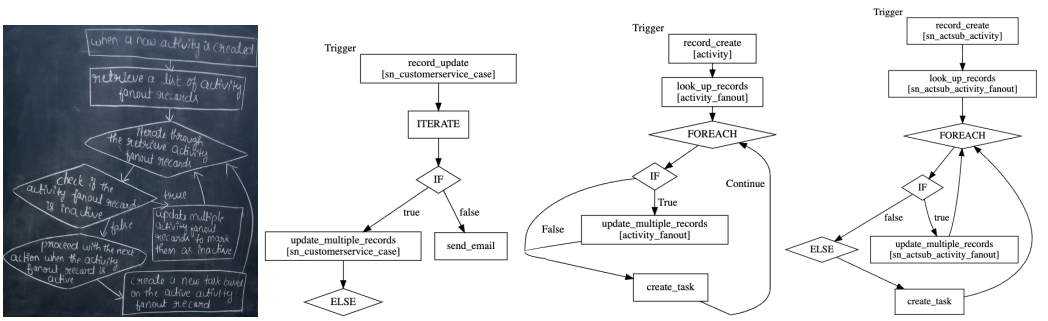
We are interested in the capability of our models to generalize to out-of-distribution settings. Specifically, we fine-tuned Llama 3.2 11B Vision Instruct and Pixtral 12B exclusively on synthetically generated workflow diagrams (Synthetic) and evaluated their performance across diverse and out-of-distribution (OOD) diagram styles (see Table 3).

Results show that models trained solely on synthetic data achieve large gains over their base versions: for Llama 3.2 11B Vision Instruct, the average Flow Similarity increases from 43.5 (base) to 78.7, and for Pixtral 12B from 58.8 to 86.0. Nevertheless, fine-tuning on the full, diverse dataset yields the best results, reaching 91.9 for Llama and 91.6 for Pixtral, highlighting the continued importance of data diversity.

Table 3: Performance of models by sample type (%). Models finetuned solely on synthetic data outperform the base models in most cases. The reported metric is Flow Similarity with Inputs.

	Synt	Digi	Man	WB	UI	Avg
Llama 3.2 11B Vision Instruct						
Base	45.7	44.7	39.0	35.6	47.8	43.5
FT (Synth)	92.9	65.9	66.7	53.0	58.1	78.7
FT (All)	**94.3**	**90.0**	**88.3**	**91.0**	**95.3**	**91.9**
Pixtral 12B						
Base	59.5	57.6	54.6	55.4	71.7	58.8
FT (Synth)	92.1	86.0	80.0	79.2	58.3	86.0
FT (All)	**92.7**	**90.8**	**89.4**	**92.4**	**97.0**	**91.6**

Moreover, to better assess generalization, we evaluate models on 300 OOD samples, incorporating real-world and human-generated diagrams that vary in style and complexity. Figure 3 shows an example, originating from



(a) Ground Truth (b) Llama 3.2 11B (c) GPT-4o (d) Llama 3.2 11B (FT)

Figure 5: Blackboard sketch of a workflow with rendered workflows generated by different VLMs.

a workflow platform not represented in the training data. The OOD evaluation set reveals that our fine-tuned models substantially outperform their base counterparts, while fine-tuned Pixtral even surpasses GPT-4o and Gemini (see Figure 4). These results validate the effectiveness of our approach and highlight potential improvements with more data diversity. We also note that performance across all models is much lower than on the main dataset, indicating room for growth in quality.

5 Error Analysis and Discussion

In this section, we examine the current failure modes of various models in workflow generation. To illustrate this, we present a representative example that highlights the strengths and limitations of each approach. We will use the flow depicted in Figure 5a to compare the capabilities of each model qualitatively. For the sake of brevity, we will focus on Llama 3.2 11B, a finetuned variant of that same model, and GPT-4o.

When prompting a non-finetuned Llama 3.2 11B model to generate a flow, the model can struggle with several basic failures, such as predicting the wrong trigger for the task, and picking an unrelated table. Moreover, the model fails to use flowlogic elements properly, and hallucinates actions unrelated to the sketch, such as adding a component to send an email. Resulting flow can be seen in Figure 5b

A strong proprietary model like GPT-4o does perform qualitatively better on the task. In Figure 5c, we observe that the model is able to properly predict the trigger and most of the components without generating unrelated ones. However, we see that the model occasionally struggles with keeping track of the flow execution logic, where it omits an `ELSE` statement in the flow. The model also encounters difficulties with some fine-grained details in the flow that pertain to the component inputs, such as selecting a generic `activity` table when unsure of the appropriate name. In the above example, we see that the model falls back to using a generic `activity` table when unsure about what the name of the table should be given the provided information.

Finally, the finetuned variant of Llama 3.2 11B performs better than its counterparts on this example. The model predicts the appropriate flow execution logic along with all relevant components. It is also able to properly predict the right tables for the task as it has seen some data from the same domain during the finetuning phase. We include more qualitative examples illustrating limitations of our approach on various diagram styles, and compare outputs from multiple base and fine-tuned models in Appendix L.

6 Conclusion

In this paper, we presented StarFlow, a framework for structured workflow generation from sketch-based diagrams. By leveraging vision-language models and a diverse dataset, we demonstrated that finetuned models outperform general-purpose models at accurately translating sketches into structured workflow representations. Our experiments revealed key insights into the challenges posed by different sketch sources, orientations, and image resolutions, highlighting the importance of domain-specific training.

While our approach shows strong performance in workflow generation, future work could explore extending the methodology to a broader range of workflow visualization styles and improving ro-

bustness to handwritten annotations. Additionally, refining evaluation metrics to consider functional execution correctness could provide a more comprehensive assessment of generated workflows. Finally, augmenting models with external information via retrieval-augmented generation or function calling might help better ground the models in generating accurate information in the workflows. Overall, STARFLOW represents a step toward making workflow automation more accessible and intuitive by enabling seamless sketch-to-workflow generation.

REFERENCES

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone, 2024. *URL https://arxiv.org/abs/2404.14219*, 2024.

Pravesh Agrawal, Szymon Antoniak, Emma Bou Hanna, Baptiste Bout, Devendra Chaplot, Jessica Chudnovsky, Diogo Costa, Baudouin De Monicault, Saurabh Garg, Theophile Gervet, et al. Pixtral 12b. *arXiv preprint arXiv:2410.07073*, 2024.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022.

Anthropic. Claude 3.7 Sonnet System Card, 2024. URL `https://assets.anthropic.com/m/785e231869ea8b3b/original/claude-3-7-sonnet-system-card.pdf`. Accessed: 2025-03-04.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pp. 2425–2433, 2015.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Orlando Marquez Ayala and Patrice Béchard. Generating a low-code complete workflow via task decomposition and rag. *arXiv preprint arXiv:2412.00239*, 2024.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

Nastaran Bassamzadeh and Chhaya Methani. A comparative study of dsl code generation: Fine-tuning vs. optimized retrieval augmentation. *arXiv preprint arXiv:2407.02742*, 2024.

Patrice Béchard and Orlando Marquez Ayala. Reducing hallucination in structured outputs via retrieval-augmented generation. *arXiv preprint arXiv:2404.08189*, 2024.

Yuzhe Cai, Shaoguang Mao, Wenshan Wu, Zehua Wang, Yaobo Liang, Tao Ge, Chenfei Wu, Wang You, Ting Song, Yan Xia, et al. Low-code llm: Graphical user interface over large language models. *arXiv preprint arXiv:2304.08103*, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. Graphviz—open source graph drawing tools. In *Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers 9*, pp. 483–484. Springer, 2002.

Shengda Fan, Xin Cong, Yuepeng Fu, Zhong Zhang, Shuyan Zhang, Yuanwei Liu, Yesai Wu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Workflowllm: Enhancing workflow orchestration capability of large language models. *arXiv preprint arXiv:2411.05451*, 2024.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Quentin Carbonneaux, Taco Cohen, and Gabriel Synnaeve. Rlef: Grounding code llms in execution feedback with reinforcement learning. *arXiv preprint arXiv:2410.02089*, 2024.

Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Yi Su, Bohua Chen, Dongping Chen, Siyuan Wu, Xing Zhou, et al. Webcode2m: A real-world dataset for code generation from webpage designs. In *THE WEB CONFERENCE 2025*, 2025.

Jorge-Ivan Herrera-Camara and Tracy Hammond. Flow2code: from hand-drawn flowcharts to code execution. In *Proceedings of the Symposium on Sketch-Based Interfaces and Modeling*, pp. 1–13, 2017.

Drew A Hudson and Christopher D Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6700–6709, 2019.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*, 2024.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, et al. The stack: 3 tb of permissively licensed source code. *arXiv preprint arXiv:2211.15533*, 2022.

Marie-Anne Lachaux, Baptiste Roziere, Lowik Chanussot, and Guillaume Lample. Unsupervised translation of programming languages. *arXiv preprint arXiv:2006.03511*, 2020.

Chengpeng Li, Mingfeng Xue, Zhenru Zhang, Jiaxi Yang, Beichen Zhang, Bowen Yu, Binyuan Hui, Junyang Lin, Xiang Wang, and Dayiheng Liu. Start: Self-taught reasoner with tools. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13523–13564, 2025.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer vision–ECCV 2014: 13th European conference, zurich, Switzerland, September 6-12, 2014, proceedings, part v 13*, pp. 740–755. Springer, 2014.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916, 2023.

Zejie Liu, Xiaoyu Hu, Deyu Zhou, Lin Li, Xu Zhang, and Yanzheng Xiang. Code generation from flowcharts with texts: A benchmark dataset and an approach. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 6069–6077, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-emnlp.449. URL https://aclanthology.org/2022.findings-emnlp.449/.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*, 2021.

Microsoft. Power Automate - Microsoft Power Platform, 2025. URL https://www.microsoft.com/en-us/power-platform/products/power-automate. Accessed: 2025-03-05.

MuleSoft, a Salesforce Company. Automation with MuleSoft, 2025. URL https://www.salesforce.com/mulesoft/automation/. Accessed: 2025-03-05.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*, 2022.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318, 2002.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*, 2020.

Juan Rodriguez, Xiangru Jian, Siba Smarak Panigrahi, Tianyu Zhang, Aarash Feizi, Abhay Puri, Akshay Kalkunte, François Savard, Ahmed Masry, Shravan Nayak, et al. Bigdocs: An open and permissively-licensed dataset for training multimodal models on document and code tasks. *arXiv preprint arXiv:2412.04626*, 2024a.

Juan A. Rodriguez, Abhay Puri, Shubham Agarwal, Issam H. Laradji, Pau Rodriguez, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. Starvector: Generating scalable vector graphics code from images and text, 2024b. URL https://arxiv.org/abs/2312.11556.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

ServiceNow. Flow Designer - ServiceNow, 2025. URL https://www.servicenow.com/products/platform-flow-designer.html. Accessed: 2025-03-05.

Chufan Shi, Cheng Yang, Yaxin Liu, Bo Shui, Junjie Wang, Mohan Jing, Linran XU, Xinyu Zhu, Siheng Li, Yuxiang Zhang, Gongye Liu, Xiaomei Nie, Deng Cai, and Yujiu Yang. Chartmimic: Evaluating LMM's cross-modal reasoning capability via chart-to-code generation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=sGpCzsfd1K.

Shreya Shukla, Prajwal Gatti, Yogesh Kumar, Vikash Yadav, and Anand Mishra. Towards making flowchart images machine interpretable. In *International Conference on Document Analysis and Recognition*, pp. 505–521. Springer, 2023.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Peter Tong, Ellis Brown, Penghao Wu, Sanghyun Woo, Adithya Jairam Vedagiri IYER, Sai Charitha Akula, Shusheng Yang, Jihan Yang, Manoj Middepogu, Ziteng Wang, et al. Cambrian-1: A fully open, vision-centric exploration of multimodal llms. *Advances in Neural Information Processing Systems*, 37:87310–87356, 2025.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3156–3164, 2015.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024.

Weixiang Yan, Yuchen Tian, Yunzhe Li, Qian Chen, and Wen Wang. Codetransocean: A comprehensive multilingual benchmark for code translation. *arXiv preprint arXiv:2310.04951*, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9556–9567, 2024.

Zhen Zeng, William Watson, Nicole Cho, Saba Rahimi, Shayleen Reynolds, Tucker Balch, and Manuela Veloso. Flowmind: automatic workflow generation with llms. In *Proceedings of the Fourth ACM International Conference on AI in Finance*, pp. 73–81, 2023.

Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing*, 18(6):1245–1262, 1989.

Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y Wu, Yukun Li, Huazuo Gao, Shirong Ma, et al. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *arXiv preprint arXiv:2406.11931*, 2024.

A EXAMPLE JSON FROM THE WORKFLOW GENERATION HEURISTIC


```
{
  "type": "flow",
  "scope": "global",
  "trigger": {
    "annotation": "on wednesdays at a quarter to 5 pm",
    "type": "weekly",
    "inputs": [
      {
        "name": "day_of_week",
        "value": "3"
      },
      {
        "name": "time",
        "value": "1970-01-01 16:45:00"
      }
    ]
  },
  "components": [
    {
      "annotation": "look up incident tasks",
      "category": "action",
      "definition": "look_up_records",
      "scope": "global",
      "order": 1,
      "inputs": [
        {
          "name": "table",
          "value": "incident_task"
        }
      ]
    },
    {
      "annotation": "for all",
      "category": "flowlogic",
      "definition": "FOREACH",
      "scope": "global",
      "order": 2,
      "inputs": [
        {
          "name": "items",
          "value": "{{1.Records}}"
        }
      ]
    },
    {
      "annotation": "if the task is inactive",
      "category": "flowlogic",
      "definition": "IF",
      "scope": "global",
      "order": 3,
      "block": 2,
      "inputs": [
        {
          "name": "condition",
          "value": "{{2.item.active}}=false"
        }
      ]
    },
    {
      "annotation": "post incident details on MS Teams",
      "category": "action",
      "definition": "post_incident_details",
      "scope": "sn_ms_teams_ah",
      "order": 4,
      "block": 3
    }
  ]
}
```


Figure 6: **Example of a flow generated by the Scheduled Loop heuristic.** The flow includes a scheduled trigger, lookup action, conditional logic, and an MS Teams action.

B Additional Ablations

B.1 Orientation of Sample

Workflow diagrams can be represented horizontally or vertically without changing meaning. As such, we are interested in assessing whether models are better at interpreting sketches presented top-to-bottom (portrait) versus left-to-right (landscape). Our criteria for differentiating the two sample types is based on the aspect ratio of the image. We define samples where images are twice as wide as tall as *landscape*, and the rest as *portrait*.

Our results, summarized in Figure 7, show that all benchmarked models exhibit a slight drop in performance and that this gap is more pronounced for the non-finetuned variant of Pixtral-12B (even as this gap is reduced after finetuning). We hypothesize that part of the difference in performance might be explained by the composition of each split. For example, User Interface samples, which are easier (see Section 4.4), are largely portrait samples due to the nature of the data collection. The presence of such samples in the Portrait category may partially explain the performance gap.

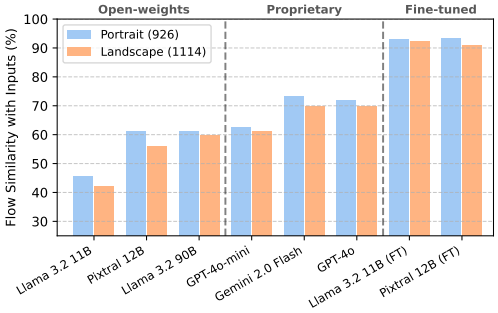


Figure 7: **Impact of image orientation**. We report the FlowSim *with input* results. Number of supporting examples for each sample type is shown in parentheses.

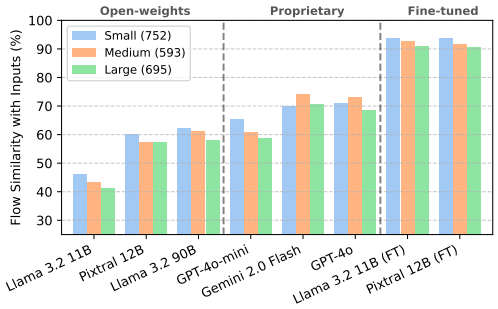


Figure 8: **Impact of image resolution**. We report the FlowSim *with input* results. The number of supporting examples for each sample type is shown in parentheses.

B.2 Ablation on Image Resolution

We study the effect of image resolution on model performance. We split sample images into three categories based on size: small (less than 400k pixels), large (more than 1M pixels), and medium (in between). We choose these image sizes as boundaries to ensure categories are approximately the same size. We present results in Figure 8.

We observe that GPT-4o and Gemini-2.0-Flash perform better on samples of medium size compared to smaller or larger samples by a non-negligible margin. This trend does not repeat for open-weight models other models as they both perform better in smaller images. It is worth noting that Pixtral's performance remains stable for larger images, whereas LLaMA exhibits a consistent degradation as image size increases. Finally, we observe this trend of degraded performance on larger samples remaining after finetuning, although it is to a lesser extent.

C End-to-End vs Task Decomposition

Here, we compare whether our end-to-end baseline approach of sketch to workflow generation can match the performance of a pipeline that decomposes the task into multiple subtasks. Following a methodology that closely matches Ayala & Béchard (2024), we first introduce the task of sketch to workflow summary, which aims to boil down the different actions performed in a given workflow sketch into a natural language summary. Then, we use this summary to first generate a workflow outline from the generated summary, and finally generate inputs for the trigger and each action found in the generated flow outline iteratively. This modular approach allows us to use a more sophisticated approach to sketch to workflow generation that can incorporate search calls to retrieve relevant actions and inputs to include in the final flow. For each of our experiments, we use GPT-4o

as the image summarizer and a different model for workflow generation (proprietary, open-weights, or finetuned). Results are presented in Table 4.

Model	FlowSim no input	FlowSim w/ input
Sketch → Workflow		
GPT-4o	*0.786*	*0.707*
Pixtral-12B	0.632	0.582
Pixtral-12B (ft)	**0.952**	**0.919**
Sketch → Summary → Outline → Workflow		
GPT-4o	0.727	0.647
Mistral-Nemo-Instruct-2407	0.472	0.414
Mistral-Nemo-Instruct-2407 (ft)	**0.834**	**0.828**

Table 4: **Impact of Task Decomposition on Flow Similarity.** Flow similarity scores for the end-to-end Sketch to Flow task compared to a stepwise approach that decomposes the task into subtasks (Sketch → Summary → Flow Outline → Flow with Inputs).

We find that decomposing the task into multiple subtasks yields lower results across the tested models. This is most likely due to errors compounding every step of the generation pipeline: every small detail missed by the summarization step will impact the generation of the flow outline, which will itself impact which inputs get populated for each component. Moreover, keeping the task of image to flow generation as a single task can substantially decrease the total latency of the application as the number of total calls to the LLM or VLM are significantly reduced.

D Human Annotators

We partnered with a for-profit data labeling company (referred to as the "Vendor") specializing in data curation for AI applications. The annotation process spanned a three-month period, beginning with a pilot phase in the first month. During this phase, we collaborated closely with the Vendor's annotation team, conducting detailed reviews and providing extensive feedback to ensure annotators fully understood the task requirements.

Our dataset was annotated by a dedicated team of 24 professionals based in India. These annotators possessed strong proficiency in technical writing and English, with educational backgrounds primarily in engineering, computer science, and related disciplines. The majority held bachelor's degrees, while some had advanced degrees in specialized fields. Additionally, they brought prior experience in data labeling, ensuring familiarity with structured annotation tasks.

To ensure the highest standards of annotation quality, a comprehensive quality assurance framework was implemented, requiring each annotation to undergo at least three independent review stages. The process began with an initial annotation conducted by experienced annotators or trainers, followed by a primary quality assurance review, where a specialist assessed accuracy, completeness, and adherence to annotation guidelines. Finally, a secondary review ensured consistency and alignment with evolving project requirements. This structured, multi-tiered approach reinforced annotation quality, minimized inconsistencies, and enhanced dataset reliability.

To uphold ethical labor standards and maintain high annotation quality, all annotators were compensated at rates exceeding fair market wages in their respective countries. This strategy supports the recruitment and retention of highly skilled professionals, fostering long-term engagement and ensuring annotation consistency across the project.

Content Safety. Annotators were instructed not to include any real names, emails, IDs, or screenshots of live systems. All hand-drawn images depict synthetic entities. A three-stage QA (initial review, primary QA, secondary QA) rejected any samples containing potentially identifying text. For UI-rendered samples, flows were generated from synthetic seeds only. We ran an OCR pass to spot obvious PII tokens prior to packaging.

E TRAINING DETAILS

We use a consistent training setup for all finetuned models presented in this paper. To mitigate overfitting, we applied early stopping based on evaluation loss. The learning rate was initialized at 2×10^{-5}, and we used the AdamW optimizer (Loshchilov & Hutter, 2017) with β values of $(0.95, 0.999)$, weight decay of 1×10^{-6}, and an epsilon value of 1×10^{-8} to ensure numerical stability. The learning rate followed a cosine schedule with a warmup phase of 30 steps. Additionally, we enforced a maximum gradient norm of 1.0 to prevent gradient explosion.

For all finetuning runs, we trained both the language model and the connector components of the VLM while keeping the vision encoder frozen. Each model was trained to support sequences of up to 32k tokens, including both image and text inputs.

We conducted training using 16 NVIDIA H100 80GB GPUs across two nodes. Full Sharded Data Parallel (FSDP) (Rajbhandari et al., 2020) was employed without CPU offloading. We also used mixed-precision training with bfloat16 (`bf16`).

F POTENTIAL RISKS

While STARFLOW offers a promising step toward automating workflow generation, it is crucial to emphasize that automatically generated workflows should never be executed directly in production environments. Generated flows may contain logical errors, unsafe configurations, or unintended actions due to model hallucinations or misinterpretation of visual inputs.

To mitigate these risks, all workflows must be reviewed and verified by human experts prior to deployment. We strongly recommend that models be integrated within sandboxed or staging environments that allow thorough functional testing and validation of every action and trigger. Automated safeguards should prevent execution on live systems without explicit human approval.

In future iterations, we plan to include automated static and dynamic validation checks to detect potentially unsafe or destructive actions before execution, further reinforcing the human-in-the-loop principle that underpins responsible workflow automation.

G ETHICAL STATEMENT

Automatically generated workflows may contain logical errors, unsafe configurations, or unintended actions due to model misinterpretation or hallucination. All generated workflows must be reviewed by humans and exercised first in a sandbox/staging environment with guardrails that block destructive operations (e.g., writes to production tables, external API calls). We recommend static checks (schema/permission validation) and dynamic tests (dry-runs with synthetic data) prior to any deployment. Execution in live systems should require explicit human approval.

Artifact Use Consistency. All external artifacts (open-weight VLMs and libraries) were used within their stated research licenses/terms. For STARFLOW artifacts, we specify research-only intended use; derivatives must comply with the original access conditions and must not be deployed in production automations without human oversight and sandbox validation.

Licensing & Intended Use. The STARFLOW dataset and code are released under the Apache-2.0 license. Finetuned model checkpoints inherit the license and usage terms of their respective base models; users must comply with those upstream licenses when using or redistributing our finetuned weights.

Use of AI Assistants. We used AI assistants solely for wording/grammar suggestions on the manuscript draft; no content, code, data, or analysis was generated without human verification.

H LIMITATIONS

While STARFLOW demonstrates strong performance in translating workflow sketches into structured outputs, several limitations remain. First, our models rely heavily on the diversity and fidelity of the training dataset. Although we curated synthetic, human-drawn, and interface-based diagrams, the coverage of real-world workflow styles and enterprise-specific notations is still incomplete, which may affect generalization to unseen diagram conventions. This issue is amplified in out-of-distribution scenarios: as shown in our OOD experiments, diagrams that diverge substantially from the training distribution (in layout, annotation style, or component positioning) lead to increased structural and semantic errors.

Second, evaluation is primarily based on structural similarity metrics (FlowSim, TreeBLEU, Trigger/Component Match). These metrics assess syntactic alignment but do not capture whether the generated workflow would execute correctly or satisfy user intent. Further, the current evaluation protocol does not measure the frequency or severity of hallucinated components. Future work should quantify hallucinations explicitly and incorporate execution-based or behavior-level evaluation (e.g. as seen in Béchard & Ayala (2024); Bassamzadeh & Methani (2024)).

Third, despite substantial improvements from finetuning, the models still struggle to ground generation to real-time information. For example, verifying whether a referenced component or table actually exists in the environment remains a challenge. This limitation sometimes leads to the creation of invalid or unused components. Stronger grounding mechanisms (e.g., retrieval-augmented generation with access to API schemas or component registries) could help mitigate this.

Fourth, our approach remains sensitive to noisy inputs such as messy handwriting, cluttered diagrams, or ambiguous component names, suggesting a need for more robust integrating of visual parsing with semantic priors or external knowledge. Finally, finetuning large vision–language models incurs substantial computational cost, limiting accessibility in low-resource settings.

Future work could address these challenges by expanding dataset coverage with more heterogeneous, real-world samples, incorporating execution-based evaluation metrics, explicitly tracking component hallucination rates, and exploring retrieval-augmented or tool-assisted generation to improve grounding, generalization, and computational efficiency.

Compute & variance. Due to compute constraints, some results are from a single seed; we therefore provide scripts to reproduce runs. Future work will widen the seed sweep and include confidence intervals.

I FUTURE DIRECTIONS

Building on the limitations identified above, several promising research directions emerge. First, future work should explore execution-based evaluation of generated workflows (e.g. Chen et al. (2021); Austin et al. (2021). While current metrics assess structural similarity, they do not capture whether the synthesized workflow functions as intended during runtime. Integrating simulation or sandboxed execution environments would enable semantic validation of control flow, component compatibility, and real-world task completion.

Second, agentic workflow generation represents a natural evolution beyond static sketch translation. Rather than producing workflows in a single forward pass, models could be embedded within reasoning-enabled frameworks (e.g., ReAct-style agents Yao et al. (2022)) capable of tool invocation (Schick et al., 2023; Li et al., 2025), and self-correction (Gehring et al., 2024). Such agents could query documentation, perform component lookup, verify field or table availability, and incorporate real-time constraints before finalizing the generated flow. This hybrid approach would improve grounding and reduce hallucination.

Overall, these directions shift workflow generation from static prediction toward **interactive, execution-aware, and tool-augmented agents**, enabling robust automation and iterative co-design of flows with human operators.

J Decomposition of a Workflow into its Tree Representation

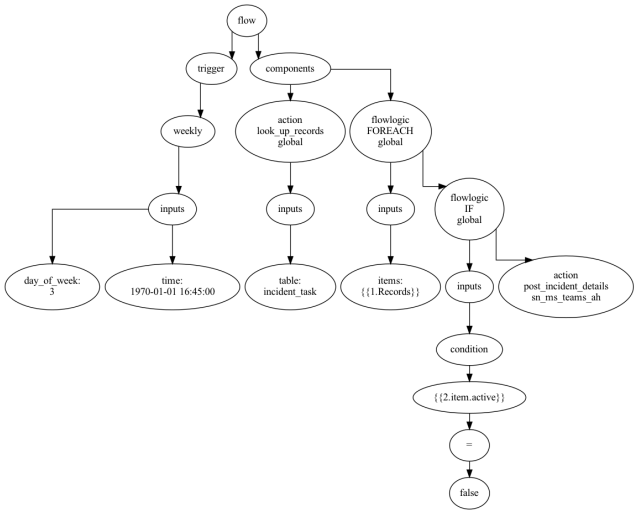


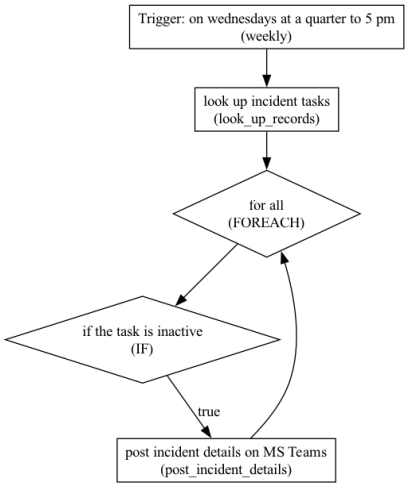
Figure 9: Decomposition of a workflow into its tree structure. This representation is used to compute Flow Similarity and TreeBLEU metrics.

K Types of Workflow Diagram Samples

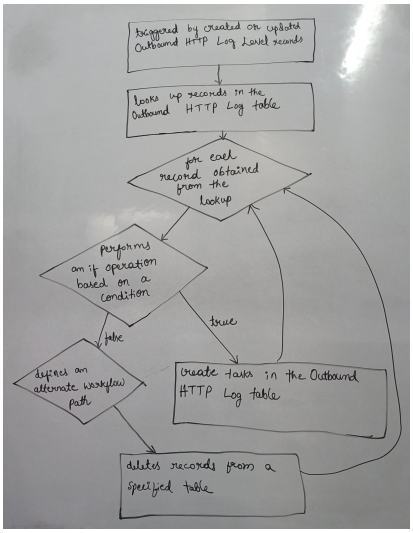
Figure 10 contains examples of the different sample types used for training and evaluation of our models.

L More Error Analysis

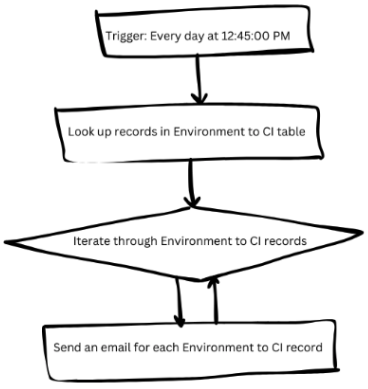
Figures 11 and 12 present more error failure modes of different models on various types of sample.



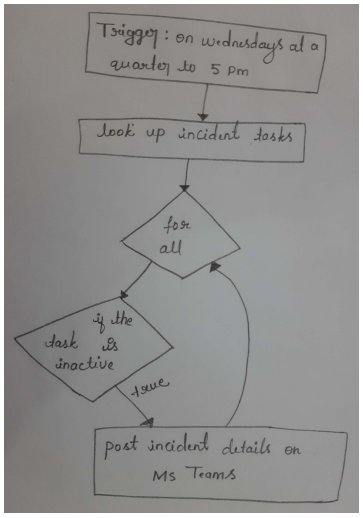
(a) Graph-based representation of a workflow (SYNTHETIC).



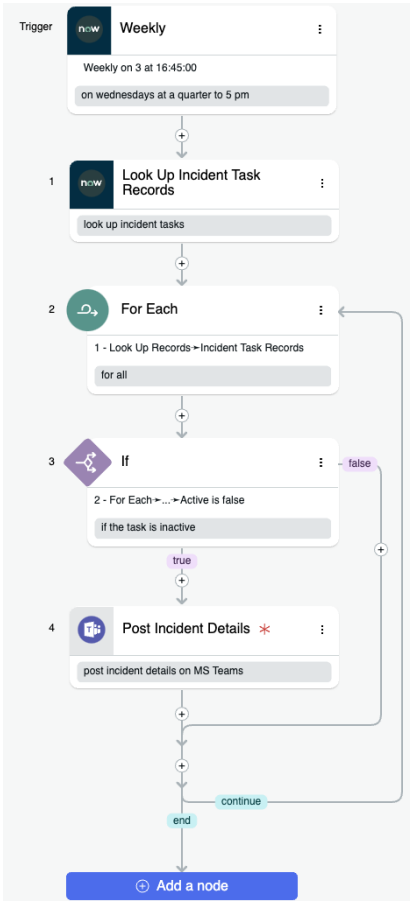
(d) Hand-drawn workflow on a whiteboard (WHITEBOARD).



(b) Digitally drawn workflow sketch (DIGITAL).

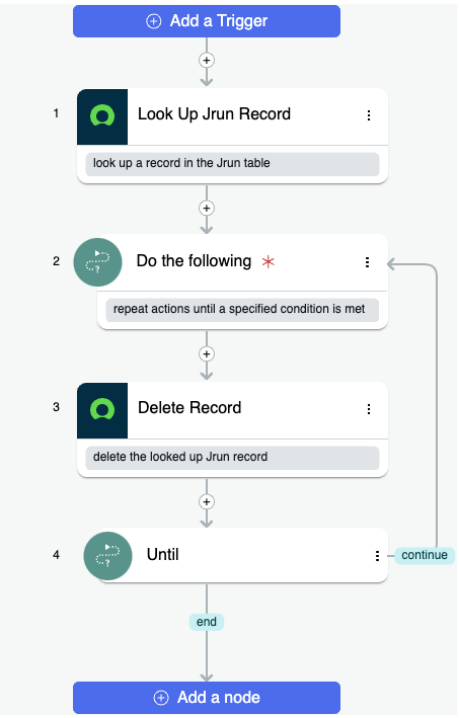


(c) Hand-drawn workflow on paper (MANUAL).

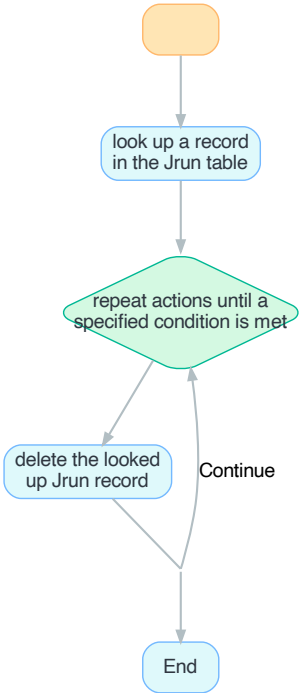


(e) Workflow rendered in the ServiceNow UI (USER INTERFACE).

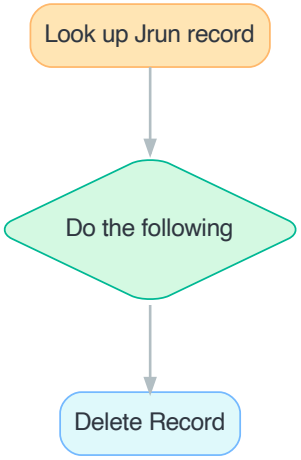
Figure 10: **Types of workflow diagram samples.** We show representative examples from each data source used in our dataset: synthetic graph-based workflows, digitally drawn sketches, manual paper sketches, whiteboard sketches, and workflows rendered directly in a user interface.



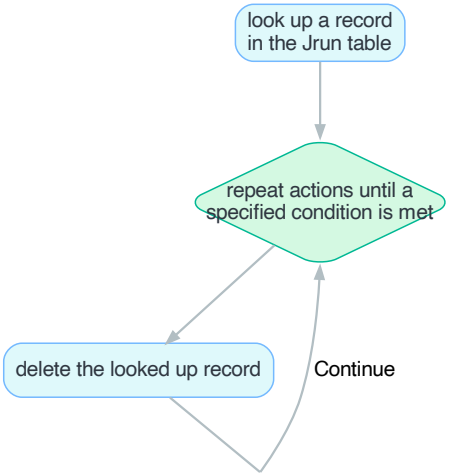
(a) USER INTERFACE Workflow Diagram



(b) GPT-4o

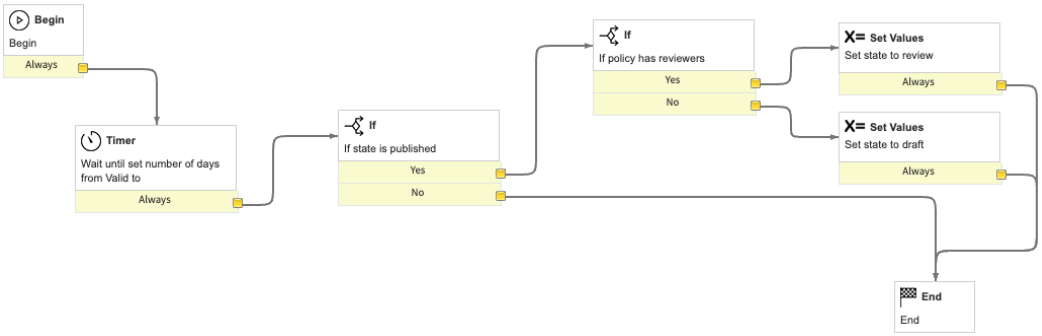


(c) Qwen-2.5-VL-7B-Instruct

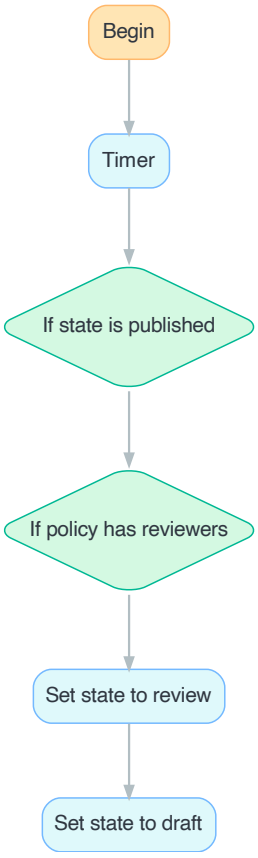


(d) Qwen-2.5-VL-7B-Instruct (Finetuned)

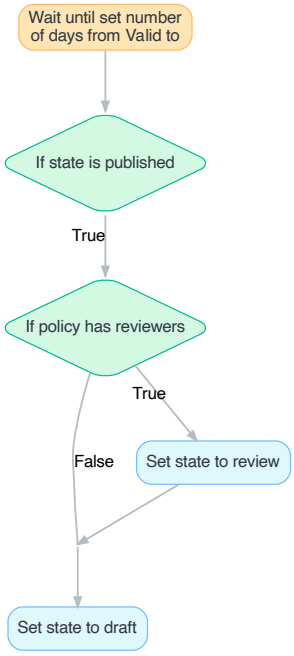
Figure 11: **Error analysis across base and fine-tuned models on a USER INTERFACE workflow diagram.** The Qwen-2.5-VL-Instruct model exhibits structural misinterpretations, confusing a `DOUNTIL` loop with an `IF` condition and incorrectly treating the `look_up_records` element as a trigger rather than a component. GPT-4o captures the correct flow logic but introduces an unnecessary empty trigger and an extra `end` component; it also selects an incorrect input table (not illustrated in the diagram). In contrast, the fine-tuned Qwen-2.5-VL model generates a fully accurate workflow, correctly handling both control logic and component definitions.



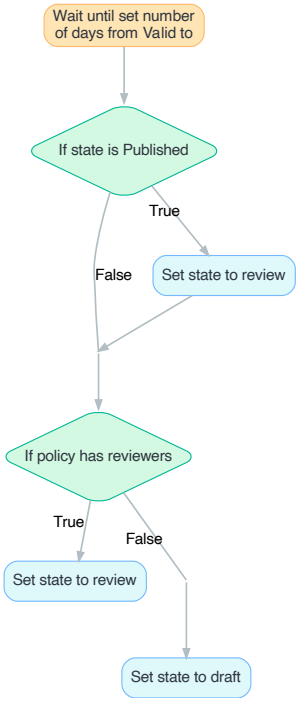
(a) OUT-OF-DISTRIBUTION Workflow Diagram



(b) Pixtral-12B



(c) GPT-4o



(d) Pixtral-12B (Finetuned)

Figure 12: **Error analysis across base and fine-tuned models on an out-of-distribution workflow diagram.** The Pixtral 12B model introduces a redundant `begin` trigger, incorrectly interprets the `IF` statements resulting in invalid control logic, and hallucinates `set_values` components instead of generating the expected `update_record` actions. GPT-4o produces a mostly correct flow but misclassifies the `TIMER` component as a trigger, and exhibits slightly incorrect logic in the second conditional, causing all records to be assigned the draft state while also hallucinating `set_state` components instead of `update_record` (not shown in diagram). The fine-tuned Pixtral 12B model correctly reflects condition-dependent record updates but still misidentifies `TIMER` component as a trigger and fails to terminate the flow when the state is not published.