See, Record, Do: Automated Generation of UI Workflows from Tutorial Videos

Adam Beauchaine
Worcester Polytechnic Institute
100 Institute Road, Worcester, MA
ajbeauchaine@wpi.edu

Craig Shue
Worcester Polytechnic Institute
100 Institute Road, Worcester, MA
cshue@wpi.edu

Abstract

System designers and developers need data-driven approaches for user-interface (UI) development and testing. They need trace-based workflows to support UI navigation agents and inputs for UI code generation. Given the high production costs of manually constructing such workflows, the UI agent research community has explored automated, fully-synthetic UI workflow generation [16]. However, there is an open need to characterize the fidelity and effectiveness of these synthetic approaches with respect to the current manual approaches.

In this work, we aim to provide larger-scale synthetic workflows based on past human usage. We particularly focused on the desktop application modality since prior synthetic generation has mostly targeted mobile or web applications. Using video tutorials with permissive licenses, we derive associated UI behaviors to construct synthetic dataset of UI workflows in desktop applications. We provide data from these videos to a large language model (LLM) to generate a set of "task list" instructions that replicate the videos' actions. We execute the task list instructions in a UI agent within an instrumented desktop operating system. Using the sensor data from that environment, we can enable simpler actuation scripts for replay of the tasks. Along with detailing our approach, we provide a dataset with over 5,000 workflows across a range of desktop applications with costs that are significantly lower than manual construction.

1. Introduction

Data-driven approaches are important for a range of development and automation approaches in a computer's user interface. Software designers can significantly improve design process for the user experience (UX) using approaches such as automatic code generation [32]. For software testing or execution purposes, UI navigation agents [38] can perform tasks within graphical UI systems. However, Chen et al. [4] note a lack of high quality user interface workflow data for desktop applications. The most prolific datasets, such as RICO [7] and Android in the Wild [28], are for mobile user interfaces. Recent work, such as the work by Xu et al. [36], has focused on web interfaces.

Unfortunately, the generation of desktop UI workflow data faces multiple challenges. Existing approaches that leverage crowdwork for UI datasets may be prohibitively expensive for organizations and individuals. The construction of the RICO data set cost over $19,000 USD and required over a dozen human workers to complete [7]. With the rise of large language model (LLM)-based UI agent systems, we explore the extent such agents can reduce the time and resource costs associated with UI data generation.

To the best of our knowledge, only Xu et al. [36] have explored LLM UI agents for workflow generation. In their work, the authors collect text-based online tutorials of web applications for replication via a UI-Agent. While there have been concerns raised about the utility of generative approaches regarding UI data [16], the use of such UI agents as synthetic data sources for UI workflows is not well understood. There is a clear need for a characterization of both the costs, requirements, and fidelity of synthetic data generation of UI workflows via LLM-based UI agents.

To address these needs, we construct a high-fidelity dataset comprised of user interaction traces of popular desktop applications, totaling to over 5,000 unique workflows. We define a scope of popular applications in the most ubiquitous desktop operating system, Microsoft Windows. We leverage publicly-available tutorial video resources and an LLM to generate user-inspired task lists. We supply these task lists to a UI agent for reconstruction and recording, allowing for the automated generation of user workflow data.

Our approach has two key distinctions from prior work. First, we leverage multimodal data in the form of Creative Commons YouTube videos as our primary data source, which allows for the inclusion a visual element that has not been studied before. In addition, we expand the actuation and recording of workflows to include all applications on a desktop environment, as opposed to strictly browser-based approaches. We present performance measurements to characterize hardware costs and generational throughput. We characterize the input requirements for generating UI data through LLM-based UI agents. By leveraging metrics to evaluate the fidelity and distinctiveness of generated workflows, we additionally evaluate the success rate of LLMs at completing user-generated tutorial tasks.

The primary costs associated with this strategy are related to the nature of LLM-based UI agents. If one does not have the capacity to host these inference models locally, the API costs for workflow generation may be expensive for some models. Further, while current models offer improved performance in popular applications, the failure rate of these approaches may be high. This requires users to filter through poorly-performing workflows to attain high quality data by our metrics.

Competitive approaches lack the modality focus [7, 28] or scale [30] of our approach. Our work features similarities to Xu et al. [36], but we shift our focus to desktop applications as opposed to web applications, and provided detailed performance-based metrics. We additionally expand the actuation goals from a single text-based goal to an entire task list within an application.

In the process of constructing this dataset, we ask the following research questions:

1. To what extent can we create and actuate synthetic user interface task lists from visual observations of application workflows?
2. What are the data requirements and associated costs for generating synthetic user interface workflows?
3. What are the performance costs associated with large-scale workflow actuation using GUI agents?

To explore these research questions, we leverage a combination of open source tools as well as our own developed strategies. We construct and evaluate our dataset production strategies as well as outputs. In so doing, we make the following research contributions:

1. A high-fidelity, synthetic user workflow dataset with real actuation data gathered from LLM-brained GUI agents
2. An evaluation of data gathering and procedural strategies for generating synthetic workflow data
3. A formal construction and evaluation of the actionability of visual observations from software usage videos.

In Section 2, we review relevant background material involving user interfaces and prior dataset construction. We detail our own design work and formalism for UI task analysis in Section 3. In Section 4, we outline our approach to our pipeline tools. We discuss the detailed results and conclusions of our work in Sections 5 and 6.

2. Background and Related Work

We characterize the current state of UI-focused datasets along with the tool-sets we utilize in our work. We summarize the existing state of user workflow datasets across a range of modalities, as well as the usage of synthetic data in UI trajectory research.

	Ours	**AgentTrek**	**GUIAct**	**Mind2Web**	**WebLINX**
Workflows	**5,040**	10,328	2482	1009	969
Desktop	✓	✗	✗	✗	✗
DOM Trees	✓	✓	✗	✗	✗
Multi-App	✓	✗	✗	✗	✗
Reasoning	✓	✓	✗	✗	✗
Screenshot	✓	✓	✓	✗	✓

Table 1. A comparison with existing workflow-based datasets.

2.1. User Interface Workflow Datasets

Since graphical UIs are a common way for users to control applications, they have become the subject of efforts to automate interactions. Early approaches for graphical UI testing often leveraged controlled explorations of UI states within desktop [34] and mobile [14] applications. In the desktop space, such efforts have accelerated due to the development of Microsoft UIAutomation [6] and similar tools for providing formal labeling of graphical elements.

Prior work has sought to produce data-driven approaches related to interface design and testing. These efforts often consist of large repositories of user interface runtime data, typically extracted during a user workflow on a device [7]. Such data can provide insight into both trends of UI design, as well as offer valuable training data for the automated exploration and fine-tuning of UI automation models [19]. Deka et al. [7] were the first to record UI data at a significant scale. In that work, the authors leverage crowd-workers to automate exploration processes of Android applications. Rawles et al. [28] employ a similar approach on Android devices with greater application diversity.

2.2. User Interface Agents and Benchmarking

We use a general definition of a UI Agent as any form of software capable of actuating workflows within graphical UI systems through the emulation of human atomic actions such as clicking and typing. While early agents leveraged record and replay [15] or reinforcement learning solutions [10], LLM-based methods emerged recently with higher performance [23]. This improvement comes from the innate reasoning abilities of new LLMs [40] as well as the capacity for increased performance via fine-tuning [9].

LLM-based agents often leverage vision models in combination with real-time input data to actuate workflows in a graphical environment. Works such as Li et al. [17] and Burns et al. [2] seek to enhance vision language model capabilities through the mapping of natural language phrases to graphical UI frames. Wu et al. [35] use such data to create a foundation model for UI agent navigation tasks that is competitive with much larger models such as GPT4o.

App Category	Application	Included Workflows	Avg Len	Min Len	Max Len	Complexity
Office	Adobe Acrobat	24	8.67	4	16	Medium
	LibreOffice Calc	61	8.72	1	25	High
	LibreOffice Impress	16	11.44	4	24	Medium
	LibreOffice Math	3	9.33	5	13	Medium
	LibreOffice Writer	70	9.69	1	20	Medium
	Microsoft Office Excel	1,407	7.27	2	33	High
	Microsoft Office PowerPoint	1,260	9.32	1	36	Medium
	Microsoft Office Teams	122	8.89	1	30	Medium
	Microsoft Office Word	278	7.87	2	29	Medium
Email	Microsoft Outlook	83	8.12	1	18	Medium
	Mozilla Thunderbird	26	7.58	1	17	Medium
Creativity	Adobe Dreamweaver	28	8.43	5	15	High
	Adobe Illustrator	566	9.35	1	32	High
	Adobe Photoshop	439	8.70	1	30	High
	Audacity	246	8.04	4	24	Medium
	LibreOffice Draw	10	8.71	4	14	Medium
Note-Taking	Microsoft OneNote	40	8.30	2	13	Medium
	Notepad	22	7.77	1	22	Low
	Notepad++	46	8.22	1	13	Medium
	Obsidian Notes	9	8.67	3	14	Medium
Web-Browsers	Google Chrome	223	6.86	1	17	Medium
	Mozilla Firefox	61	7.48	2	23	Medium

Table 2. We provide application metrics in category groups, including the number of obtained workflows, the number of tasks in each, and a coarse complexity rating (High/Medium/Low) based on keystroke-level modeling averages of applications.

With the broader use of such agents, the community has worked to formalize the evaluation of agent performance for various domains via benchmarking. Such benchmarks serve as performance indicators and fine-tuning tools for UI agents. World of Bits (WOB) [31] and successors such as MiniWoB++ [18] created a standardized set of tasks for UI agents to complete. Within benchmarking approaches, manual annotation is common and may range in scale from several researchers to whole organizations [22, 31]. Crowdwork has also played a significant role in both dataset and benchmark construction [7, 22]. We aim to construct such labeled workflows without the time-consuming manual labeling requirements of existing work.

2.3. LLMs and Synthetic Data in UI Research

Synthetic data broadly refers to annotated data generated by computer algorithms or simulations [20]. Within the HCI research community, synthetic data generation has been viewed as a potential use-case of generative AI [29]. The privacy-preservation component of synthetic data has been widely noted [20], and has allowed for accessibility-based applications in user interface design. Peng et al. [26] leverage synthetic images for code generation. In the trajectory space, Xu et al. [36] leverage public tutorial repositories to create workflows based on natural language tasks. Our own work expands these efforts to encompass multimodal workflow data for Windows desktop applications.

3. Design Goals and Actionability Formalism

When viewed through the model view controller (MVC) software architecture pattern, our work focuses on the "view" component of applications. Within the paradigm of UI interaction tasks, agent-based approaches are typically understood as partially-observable Markov decision processes (POMDP). This is due to the non-determinism of UI actions with respect to underlying system state. We thus define the notion of an application's "**observable state**" as an equivalence class:

$$q = \{s \in \mathbb{S} \mid \mathrm{sim}(s, s_g) \geq \theta\} \qquad (1)$$

In this equation, s denotes a visual representation of the UI, out of the set $\mathbb{S}$ of all possible visual states. The function sim represents a visual similarity metric between two states, in this case s and a goal state s_g. If this value is below a given threshold function θ, the two states are considered "observationally equivalent," and members of the same equivalence class q.

With this understanding of visual state in applications, we consider a natural language task list to be "actionable" if and only if it is comprised solely of actionable sub-tasks. An actionable sub-task must have the following schema:

- **Target Element:** The UI element(s) effected by the current sub-task, which are denoted by naming convention or on-screen pixel location.

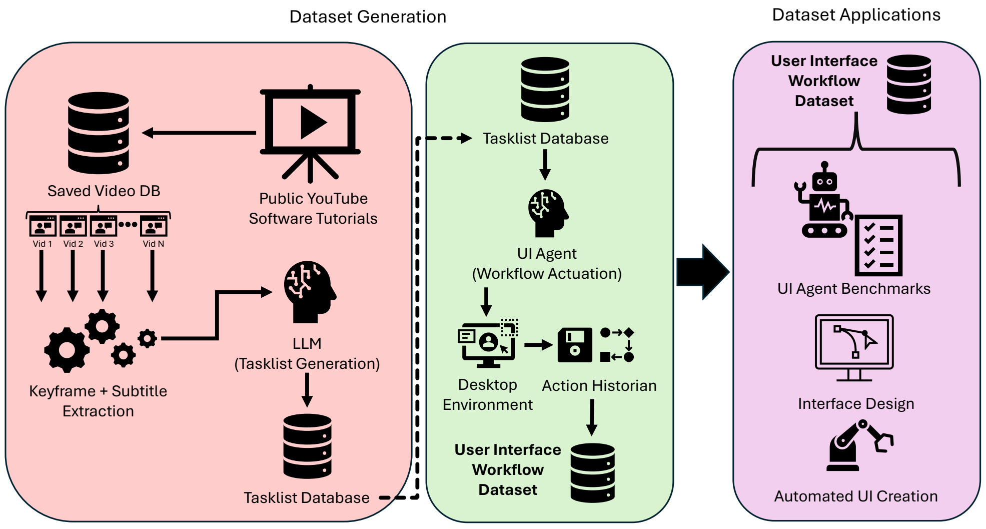

Figure 1. Overview of pipeline and applications of our work.

- **Associated Atomic Action(s):** The primitive actions (click, type, scroll) associated with the sub-task.
- **Starting State:** The expected observable state prior to sub-task actuation.
- **Goal State:** The expected observable state upon completing the sub-task actuation.

Due to the usage of natural language for UI-based task actuation, schema elements may be relative to current context (e.g., "change the current window size"), inferred for certain sub-tasks, or ambiguous (and thus unusable in automation). In light of these concerns, robust evaluation of task list outcomes are necessary when using non-deterministic generative methods, such as LLMs. We provide such outcome-based analysis in Section 5.1.

4. Approach

We detail our overall system architecture for the collection, processing, and actuation of UI workflows.

4.1. System Architecture

We develop a pipeline with UI agents being the primary generators of synthetic workflows. We start our pipeline approach by collecting public videos of application tutorials from Google's YouTube platform. We use videos with Creative Commons [5] licensing and select videos on a per-application basis. For each video, we extract subtitles and significant key-frames to produce a set of input data for summarization. We then present this data to an LLM and ask it to summarize the video into a simple workflow that may be actuated by an end user. We refer to such instructional output of the LLM as a task list.

After we create this task list, we deploy a graphical UI desktop environment equipped with a UI navigation agent. We configure each environment to have an installed copy of the software associated with the task list. We then prompt the agent to actuate the task list instructions for that application. We record screenshots, UI accessibility tree data, interaction element locations, as well as LLM context for each action taken. This recording results in a finished multimodal dataset of desktop application workflows.

4.2. Public Video Collection

At a high level, we collect the public tutorial videos using open-source libraries and toolkits. We use the Youtube API [11] and the YT-DLP library [37] to obtain videos. We use OpenAI Whisper [27] and the ResNet image detection dataset [12] and extend these tools to enable large-scale video annotation for applications. In Figure 2, we illustrate this video collection and data extraction.

We begin by conducting large scale, automated queries of YouTube videos using Google's YouTube API. When conducting queries, we utilize the "video license" tag on YouTube API requests to filter for videos under a Creative Commons license only to ensure ethical data retrieval. Where we publicly share our data set, we provide authorship attribution for the videos and links to the original sources [1]. For each downloaded video, we attempt to perform subtitle generation using OpenAI Whisper. Some videos are not in English or contain no spoken dialogue, so we may not have subtitles for all videos. We retain such videos for further processing, but without a subtitle file.

4.3. Semantic Task List Generation

To transform visual data into an actionable task list, we employ a multimodal LLM capable of analyzing both image and subtitle data extracted from our collected videos. We host this LLM locally via an Ollama server [25] with the most recent variant of Google's Gemma model (Gemma3:12b) [33]. This model met the performance and contextual length requirements for our work.

To prepare an input video for submission to the LLM, we extract images from that video to use as additional prompt data. We perform keyframe extraction on videos to only extract high-relevance frames from a workflow. Keyframes can allow for the usage of longer videos as workflow sources, since they use require less images to summarize a video. While our approach runs the risk of excluding relevant data, UI frames are unique in that models may infer a significant amount from timestamped images. This may include the identification of novel elements whose creation was not captured in a previous keyframe, as well as new input data to existing UI elements.

Our keyframe extraction process consists of two steps: keyframe identification, and UI verification. In the first step, we leverage the `scene-detect` library [3] to establish a pixel similarity threshold between video frames. If this threshold is exceeded, we consider the frame to be a keyframe and save it as an image. After we extract the keyframes, we remove any keyframes that contain non-UI visual information. To distinguish UI frames from non-UI frames, we use a fine-tuned variant of the ResNet architecture for image classification [12], pretrained on ImageNet [8]. We adapt it to a binary classification task using a custom-labeled dataset of 2000 images extracted from our data. We found that such an approach outperforms approaches based on pixel structures by 23% in our evaluation. When keyframes are generated, we present them along with subtitle data to the LLM using the following prompt:

Task List Generation Prompt

You are an expert on user interface software and tutorials. You will be provided a list of frames extracted from a video about [program name]. Your task is to summarize this video into a single task list, containing simple, clear instructions for a user to follow to replicate the workflow demonstrated in the video. If creating such a list is impossible with the provided video, you may output: "task list creation not possible". Here is the data:

If the LLM can successfully generate a task list, we perform a final check for redundancy. In this check, we again leverage the LLM to perform a comparative analysis of the produced workflow to all other generated workflows for a given program. We ask the LLM to produce a binary output (similar/non-similar). If the LLM indicates a task list is similar to more than three existing task lists, we discard that task list. This allows us to accommodate a degree of LLM error while supporting the goal of a diverse set of task lists.

4.4. Task List Actuation and Data Collection

We automate the actuation and data harvesting of UI interactions associated with our task lists. We do so using UFO [39], an open-source UI agent designed for the Microsoft Windows operating system which has demonstrated successful task completion for desktop tasks in Windows.

We leverage UFO's "follower mode" to perform task list actuation. Follower mode enables UFO to follow a predetermined set of natural language steps as opposed to generating such a set via an agent. Such actuation is related to our definition of actionable task lists in Section 3. UFO is capable of interfacing with different types of LLMs. For our actuation, we leverage `Qwen2.5-VL-32B-Instruct`. A variant of the Qwen2-VL model, Qwen2.5-VL has been fine-tuned for agent actuation tasks and has been recommended by the developers of UFO for self-served models.

For each workflow, we record a detailed trace of UI interaction data, comprised of screenshots, action context data, and formal application tree states. We include the per-step screenshots taken by UFO, as well as click-based screenshots, in which mouse interactions with applications are highlighted. We capture the LLM context data UFO provides and export it after each session. After each action, we capture the application tree data, which is comprised of a structured tree of elements for the targeted application. We capture these elements using the Microsoft UIAutomation library for the application operated by the main UFO process. We store context data in a text file after each run.

5. Results

We present the results and observations relevant to our research questions.

5.1. Fidelity and Downstream Applications

We explore the research question: "To what extent can we create and actuate synthetic user interface task lists from visual observations of application workflows?" We detail our actionability-based self-critique method for task list filtering, as well as our results for uniqueness and reproduction success rate of our workflows.

We note that workflows can still be useful for downstream applications even if they fail to complete the given set of tasks. In particular, if a workflow includes novel exploration of an application, it may be useful for understanding visual state transitions. We define a workflow as being unique if it reaches a state that is not present in any prior workflows' list of states. We leverage the UIAutomation library output detailed in Section 4.4 to determine such uniqueness. That library provides output formally structured as a tree of UI nodes. These nodes include all UI elements such as titles, buttons and tab-items rendered in a Windows application. For each workflow, we perform an

$\mathbf{O(n)}$ walk of the nodes of the output tree of the final state, comparing node placements with all previous trees. We provide such a comparison in Figure 2.

When the UI agent completes a workflow, we utilize our collected outputs and compare them with the originally-obtained workflow from the source video. We compare the original task list with the recorded workflow images as provided by the actuation server. We use the same LLM with a fresh context window to perform this measurement using the prompt below, enabling a self critique measurement:

Actuation Success Verification Prompt

"You will be presented with a set of images representing a workflow on a computer user interface. Your task is as follows: given these images, determine if they represent the following structured text workflow:" [Task list] "If the images provided are an accurate sequence of events as described in the text, output YES, if not, output NO."

To provide a robust analysis, we include manual human validation of workflow success rates for 250 unique, randomly sampled workflows across 4 applications: Microsoft Powerpoint, Microsoft Word, Microsoft Excel, Google Chrome, and Adobe Photoshop. These applications were selected both for diversity as well as for having a high number of completed workflows at the time of writing. Review is conducted by a single member of the research team. To mitigate labeling bias, LLM outputs were not revealed to the labeler prior to the completion of their own labeling outputs. Correctness is based on the same prompt as given to the LLM labeler, and the labeler was only permitted to see the set of images present as part of the completed dataset. The conditions for success may be unclear in some cases, such as successfully navigating to a web page and attempting to actuate a web UI element that existed at the time of the source tutorial video, but not at the time of dataset construction. We leave such scenarios to be determined on a case-by-case basis by the reviewer, but recommended erring towards failure in such cases so as not to favorably bias results. In Table 3, we show the machine and human-reviewed workflows, with respective outcome uniqueness.

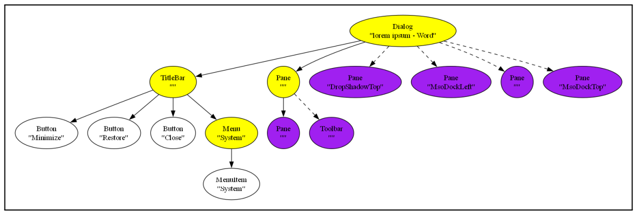

Figure 2. We compare a Microsoft Word UI tree with a prior UI tree. We do not shade identical nodes. Yellow nodes differ in content fields. Purple nodes are present only in the current tree and not in the prior one.

Application	**Hunman Eval**	**LLM Eval**	**Unique**
Excel	70%	68%	96%
Google Chrome	74%	75%	100%
Photoshop	52%	42%	84%
PowerPoint	72%	66%	98%
Word	86%	70%	96%

Table 3. Results of LLM and manual evaluation of workflow success and uniqueness of outcomes.

While application scores were variable, most applications scored highly on both the human and LLM-based correctness evaluation. All evaluated applications achieved above 70% correctness in human evaluation, with the exception of Adobe Photoshop, in which increased workflow complexity lead to a moderate reduction in completion rates. Across all workflows, unique results were high. This implies significant exploration of application visual states, even in failure scenarios.

To explore dataset applications beyond the scope of workflow success, we trained joint encoders over screenshots and UI trees from our dataset. We leverage a contrastive loss, aligning paired representations in a shared embedding space. To test cross-application generalization, we performed leave-one-out retrieval: in each case, the model was trained on two Microsoft Office applications and evaluated on the third. We select three applications in our dataset with high amounts of recorded data: Word, Powerpoint, and Excel. Queries from the held-out app were projected into the embedding space, and nearest neighbors were retrieved by cosine similarity. As shown in Figure 3, the retrieved neighbors reflect semantically consistent states (e.g. ribbons, dialogs, templates), even when the queried application was unseen during training. This demonstrates that our dataset enables generalizable UI representations that capture functional structure beyond pixels. Further, the multiple modalities captured by the dataset allow for the retrieval of structurally similar UIs with different visual appearances.

5.2. Synthetic Workflow Generation

We explore the metrics and results associated with the research question "What are the data requirements and associated costs for generating synthetic user interface workflows?" To allow for experimentation via manual annotation, we present a smaller-scope case study of five distinct applications. These applications are selected for based on the number of successful generated task lists.

The five applications we selected are Adobe Photoshop, Microsoft Outlook, Microsoft Teams, Microsoft Word, and Mozilla Firefox. We sample 100 videos for each of these applications and each video had subtitle data available. We then compare task list generation in three separate cases:

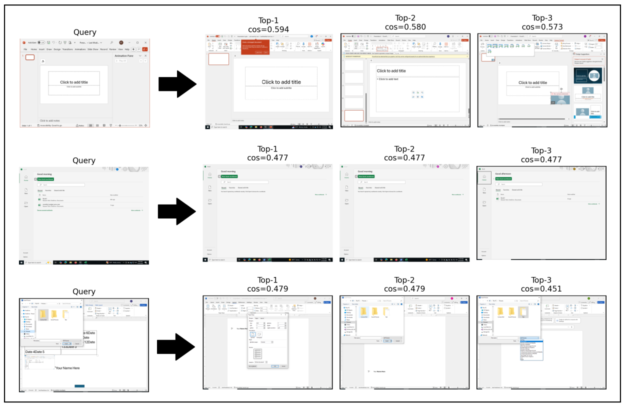

Figure 3. Zero-shot cross-application retrieval with joint image–tree embeddings. Each row queries a held-out app using an embedding trained on the other two: Row 1—PowerPoint (trained on Word+Excel); Row 2—Excel (trained on Word+PowerPoint); Row 3—Word (trained on Excel+PowerPoint).

1. **Multimodal:** We provide keyframe images and subtitles in the task list generation prompt.
2. **Text only:** We only provide the extracted subtitle data in the task list generation prompt.
3. **Image only:** We only provide the extracted keyframes from the videos in the task list generation prompt.

Across all three approaches, we achieve an overall task list yield rate of exactly 0.5 among the sampled videos. The yield rate refers to the success of the LLM in generating a task list based on the input data provided. We depict the variations in yield rates in Figure 4. We find the performance of the vision modality for Photoshop workflows to be particularly noteworthy, but understandable, given the heavy reliance of on-screen demonstrations and graphics common to artistic programs.

We then compare the relative quality of generated task lists with respect to correctness and actionability. We leverage the atomic factscore metric as proposed by Min et al. [21] in combination with our actionability schema as we described in Section 3 as a knowledge source. If a task list step is actionable, a human reviewer marks it to be supported by the knowledge source. It is then added to the total factscore [21]. We collect a sample from our case study dataset of 150 generated workflows, which we evenly select across applications and modalities. We then had a human evaluator manually review these workflows with an understanding of our actionability-based knowledge source. We display the average factscores by modality are in Table 4. We note that across all applications, the multimodal-based task lists yielded the highest factscores in comparison to vision or text-only modalities.

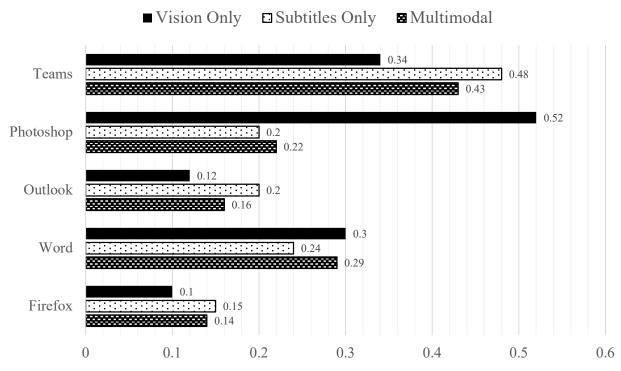

Figure 4. Task list yield rates by modality and application

Modality	Factscore	Best Application
Multimodal	0.90	Adobe Photoshop
Subtitles Only	0.82	Mozilla Firefox
Vision Only	0.79	Adobe Photoshop

Table 4. Average atomic factscores by modality

In Table 5, we show the success rates by both modality and application. While both the multimodal and subtitles-based approaches performed well across the browser (Firefox), word processor (Word), and email client (Outlook) applications, the multimodal approach significantly outperforms the subtitles-based approach in both the visual design application (Photoshop) and the enterprise communication application (Teams). Additionally, the vision and multimodal approaches significantly outperformed the subtitles approach in the visual design application. Given these results, the process of extracting task lists from videos benefits from image-based inputs across application types.

	Applications				
Modality	Firefox	Word	Outlook	Photoshop	Teams
Multimodal	0.88	0.84	0.90	0.96	0.91
Subtitles	0.92	0.86	0.81	0.79	0.72
Vision	0.71	0.85	0.80	0.97	0.66

Table 5. Atomic factscore by application

5.3. Generation and Actuation System Performance

We present the metrics and results associated with the research question: "What are the performance costs associated with large-scale workflow actuation using GUI agents?" We present hardware usage data gathered across both the LLM's task list generation and UI agent's actuation of task lists in our environment along with associated throughput and estimated monetary costs. Generation refers to the LLM-based generation of task lists based on some combination of input data as described in section 4.3. Actuation refers to the running of task lists via the UFO UI agent as described in section 4.4.

We use separate hardware environments for the task list generation and agent actuation stages in the pipeline. This allows us to achieve greater concurrency and allows us to meet the different computational requirements for these steps. In Table 6, we provide the hardware specifications of the two environments. All hardware was hosted locally in a dedicated university research environment. The LLM-UI agents required significantly higher computational resources in terms of GPUs.

Server	CPU	Memory	GPU
Generation	Gen. 13 Intel Core i9-13900K	128 GB	2x RTX 4090
Actuation	Intel Xeon Processor (Icelake)	128GB	2x H100

Table 6. Hardware Specifications of Servers

We analyze the metrics gathered from a three-hour sample run of both the generation and actuation servers. For GPU logging, we utilize the Nvidia management library system management interface (SMI) [24] to collect GPU state information at ten-second intervals. We gather CPU and memory measurements at ten-second intervals using the Linux `proc` filesystem.

During our observations, the CPU and memory utilization rates were typical of idle systems in both the actuation and generation servers. The CPU and RAM utilization of the actuation server averaged at 7.20% and 3.60% respectively. The average CPU and RAM utilization of the generation server were 16.80% and 1.20%, respectively.

We observe significantly higher resource utilization rates in the servers' GPUs. The generation server performed well and its resource utilization rates were low, even with its two consumer grade GPUs. The actuation server experienced significantly higher proportional resource usage, despite having significantly more powerful hardware. While existing UI agent systems have required powerful hardware setups, UFO's high context window requirements lead to the large memory utilization statistics seen in Figure 5.

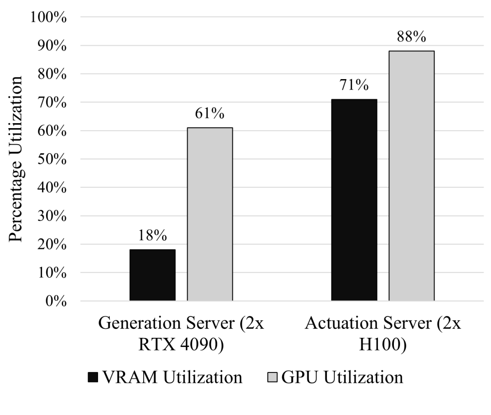

Figure 5. GPU usage across generation and actuation servers

Using computational resources at remote providers via APIs can offset local hardware requirements, but at the monetary cost of API pricing schemes and non-trivial curator time (e.g. designing prompts/instructions, onboarding, validation). In Table 7, we compare throughput rates (workflows completed/hour) and associated per-workflow costs by technique compared to crowdworkers. Our VLLM-based method can also be hosted remotely using current service architectures, so we compare these rates with standard LLM APIs. VLLM-based hosting costs are calculated assuming continuous utilization of a 2×H100 GPU setup at the current average AWS rate of $6.38/hour. The crowdwork statistics are based off of the reported usage in RICO, a dataset with comparable trace-recording and workflow complexity to our own as shown in Table 2.

Model	Cost	Throughput
VLLM (Qwen2.5)	0.91	7
API (GPT 4.1)	1.24	8
Crowdwork	1.77	—

Table 7. Average cost per workflow (USD) and completion rates (throughput) of approaches

We use data from prior studies to examine curator overhead. He et al. [13] report it took $\sim$125 seconds to onboarding workers to complete a UI-exploration task. Accordingly, we use $\sim$2 minutes per worker-application pair. Likewise, we use the value of 1 minute per workflow validated and filtered, assuming a small sample is checked as was done in RICO. We model a scenario of full GPU utilization, no API bulk discounts, and comparable workflow complexity. We find that VLLM-based hosting still offers the lowest per-workflow cost for sustained throughput, despite slightly lower average throughput. However, for small numbers of workflows, or when curator or quality-control labor is more costly, standard LLM APIs or crowdworkers may become more competitive.

6. Discussion

This work describes the production of a large-scale user interface workflow dataset across a range of applications. The produced dataset will be openly available at `https://web.cs.wpi.edu/~cshue/projects/opendata.html` after publication. We detail the pipeline and engineering strategies leveraged to produce this dataset. We provide a formalism related to the actionability of natural language UI instructions. We answer research questions associated with fidelity, diversity, solution costs. Future contributions may involve expanding the pipeline aspects of our work to additional modalities, or more standardized evaluations of downstream applications, such as UI agent training or standardized benchmarking.

References

[1] Anonymous Author(s). Public data set - due to anonymity requirements, this data set will be included only in a camera-ready submission, 2025.

[2] Andrea Burns, Deniz Arsan, Sanjna Agrawal, Ranjitha Kumar, Kate Saenko, and Bryan A Plummer. A dataset for interactive vision-language navigation with unknown command feasibility. In *European Conference on Computer Vision*, pages 312–328. Springer, 2022.

[3] Brandon Castellano and the PySceneDetect contributors. PySceneDetect: Python library and cli for video shot/scene detection. `https://github.com/Breakthrough/PySceneDetect`, March 2025. BSD-3-Clause license; supports multiple detection modes and ffmpeg integration.

[4] Dongping Chen, Yue Huang, Siyuan Wu, Jingyu Tang, Liuyi Chen, Yilin Bai, Zhigang He, Chenlong Wang, Huichi Zhou, Yiqiang Li, et al. Gui-world: A dataset for gui-oriented multimodal llm-based agents. *arXiv e-prints*, pages arXiv–2406, 2024.

[5] Creative Commons. Licenses list, 2025.

[6] Andy De George. Ui automation overview - .net framework. `https://learn.microsoft.com/en-us/dotnet/framework/ui-automation/ui-automation-overview`, Sep 2021.

[7] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th annual ACM symposium on user interface software and technology*, pages 845–854, 2017.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[9] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

[10] Juha Eskonen, Julen Kahles, and Joel Reijonen. Automating gui testing with image-based deep reinforcement learning. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pages 160–167, 2020.

[11] Google. YouTube Data API v3. `https://developers.google.com/youtube/v3/docs`, 2025.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778. IEEE, 2016.

[13] Xing He, Hansi Zhang, and Jiang Bian. User-centered design of a web-based crowdsourcing-integrated semantic text annotation tool for building a mental health knowledge base. *Journal of biomedical informatics*, 110:103571, 2020.

[14] Cuixiong Hu and Iulian Neamtiu. Automating gui testing for android applications. In *Proceedings of the 6th International Workshop on Automation of Software Test*, pages 77–83, 2011.

[15] Tristan Langer, Richard Meyes, and Tobias Meisen. Gideon replay: A library to replay interactions in web-applications. *SoftwareX*, 17:100964, 2022.

[16] Jie Li. How far can we go with synthetic user experience research? *ACM Interactions*, 31(3):26–29, 2024.

[17] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. Mapping natural language instructions to mobile ui action sequences. *arXiv preprint arXiv:2005.03776*, 2020.

[18] Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration, 2018.

[19] Guangyi Liu, Pengxiang Zhao, Liang Liu, Yaxuan Guo, Han Xiao, Weifeng Lin, Yuxiang Chai, Yue Han, Shuai Ren, Hao Wang, et al. Llm-powered gui agents in phone automation: Surveying progress and prospects. *arXiv preprint arXiv:2504.19838*, 2025.

[20] Yingzhou Lu, Minjie Shen, Huazheng Wang, Xiao Wang, Capucine van Rechem, Tianfan Fu, and Wenqi Wei. Machine learning for synthetic data generation: a review. *arXiv preprint arXiv:2302.04062*, 2023.

[21] Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Wei Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. Factscore: Fine-grained atomic evaluation of factual precision in long form text generation. *arXiv preprint arXiv:2305.14251*, 2023.

[22] Shravan Nayak, Xiangru Jian, Kevin Qinghong Lin, Juan A Rodriguez, Montek Kalsi, Rabiul Awal, Nicolas Chapados, M Tamer Özsu, Aishwarya Agrawal, David Vazquez, et al. Ui-vision: A desktop-centric gui benchmark for visual perception and interaction. *arXiv preprint arXiv:2503.15661*, 2025.

[23] Dang Nguyen, Jian Chen, Yu Wang, Gang Wu, Namyong Park, Zhengmian Hu, Hanjia Lyu, Junda Wu, Ryan Aponte, Yu Xia, et al. Gui agents: A survey. *arXiv preprint arXiv:2412.13501*, 2024.

[24] NVIDIA Corporation. NVIDIA System Management Interface (nvidia-smi). `https://developer.nvidia.com/`

`system-management-interface`, 2024. Accessed: 2025-07-18.

[25] Ollama Developers. Ollama: Run large language models locally. `https://ollama.com/`, 2024. Accessed: 2025-07-18.

[26] Yi-Hao Peng, Faria Huq, Yue Jiang, Jason Wu, Xin Yue Li, Jeffrey P. Bigham, and Amy Pavel. Dreamstruct: Understanding slides and user interfaces via synthetic data generation. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol, editors, *Computer Vision – ECCV 2024*, pages 466–485, Cham, 2025. Springer Nature Switzerland.

[27] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. *Proceedings of Machine Learning Research (PMLR)*, 2022.

[28] Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for android device control, 2023.

[29] Jingyu Shi, Rahul Jain, Hyungjun Doh, Ryo Suzuki, and Karthik Ramani. An hci-centric survey and taxonomy of human-generative-ai interactions. *arXiv preprint arXiv:2310.07127*, 2023.

[30] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, pages 3135–3144. PMLR, 2017.

[31] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3135–3144. PMLR, 06–11 Aug 2017.

[32] Åsne Stige, Efpraxia D Zamani, Patrick Mikalef, and Yuzhen Zhu. Artificial intelligence (ai) for user experience (ux) design: a systematic literature review and future research agenda. *Information Technology & People*, 37(6):2324–2352, 2024.

[33] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean-Baptiste Grill, Sabela Ramos, Édouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, . . . , Oriol Vinyals, Jeff Dean, and Demis Hassabis. Gemma 3: A multimodal, multilingual, long-context open model. Technical Report arXiv:2503.19786v1, Google DeepMind, March 2025. Model size: 12B parameters; multimodal with text and image input; 128k-token context window.

[34] Marlon Vieira, Johanne Leduc, Bill Hasling, Rajesh Subramanyan, and Juergen Kazmeier. Automation of gui testing using a model-driven approach. In *Proceedings of the 2006 international workshop on Automation of software test*, pages 9–14, 2006.

[35] Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*, 2024.

[36] Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. *arXiv preprint arXiv:2412.09605*, 2024.

[37] yt-dlp. yt-dlp: A feature-rich command-line audio/video downloader. `https://github.com/yt-dlp/yt-dlp`, 2020.

[38] Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, et al. Large language model-brained gui agents: A survey. *arXiv preprint arXiv:2411.18279*, 2024.

[39] Chaoyun Zhang, He Huang, Chiming Ni, Jian Mu, Si Qin, Shilin He, Lu Wang, Fangkai Yang, Pu Zhao, Chao Du, et al. Ufo2: The desktop agentos. *arXiv preprint arXiv:2504.14603*, 2025.

[40] Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*, 2024.