IndusGCC: A Data Benchmark and Evaluation Framework for GUI-Based General Computer Control in Industrial Automation

Xiaoran Yang[1,2,*], **Yuyang Du**[2,†], **Kexin Chen**[3,†], **Soung Chang Liew**[2,‡], **Jiamin Lu**[4], **Ziyu Guo**[3], **Xiaoyan Liu**[2], **Qun Yang**[2], **Shiqi Xu**[2], **Xingyu Fan**[3], **Yuchen Pan**[2], **Taoyong Cui**[3], **Hongyu Deng**[2], **Boris Düdder**[1], **Jianzhang Pan**[5,6], **Qun Fang**[5,6], **Pheng Ann Heng**[3,‡]

[1]Department of Computer Science, University of Copenhagen
[2]Department of Information Engineering, the Chinese University of Hong Kong
[3]Department of Computer Science Engineering, the Chinese University of Hong Kong
[4]The First Affiliated Hospital of Xiamen University, School of Medicine, Xiamen University
[5]Institute of Intelligent Chemical Manufacturing and iChemFoundry Platform, ZJU-Hangzhou Global Scientific and Technological Innovation Center
[6]Institute of Microanalytical Systems, Department of Chemistry, Zhejiang University

Corresponding Emails: soung@ie.cuhk.edu.hk, pheng@cse.cuhk.edu.hk

Abstract

As Industry 4.0 progresses, flexible manufacturing has become a cornerstone of modern industrial systems, with equipment automation playing a pivotal role. However, existing control software for industrial equipment, typically reliant on graphical user interfaces (GUIs) that require human interactions such as mouse clicks or screen touches, poses significant barriers to the adoption of code-based equipment automation. Recently, Large Language Model-based General Computer Control (LLM-GCC) has emerged as a promising approach to automate GUI-based operations. However, industrial settings pose unique challenges, including visually diverse, domain-specific interfaces and mission-critical tasks demanding high precision. This paper introduces IndusGCC, the first dataset and benchmark tailored to LLM-GCC in industrial environments, encompassing 448 real-world tasks across seven domains, from robotic arm control to production line configuration. IndusGCC features multimodal human interaction data with the equipment software, providing robust supervision for GUI-level code generation. Additionally, we propose a novel evaluation framework with functional and structural metrics to assess LLM-generated control scripts. Experimental results on mainstream LLMs demonstrate both the potential of LLM-GCC and the challenges it faces, establishing a strong foundation for future research toward fully automated factories. Our data and code are publicly available at: https://github.com/Golden-Arc/IndusGCC.

1 Introduction

As the world enters the Industry 4.0 era, system automation has become indispensable for modern factories. In complex production environments, particularly in flexible manufacturing, machines often require dynamic reconfiguration to adapt to shifting production goals, supply constraints, or real-time operational feedback. In large-scale smart factories, engineers may perform tens of thousands of

*Work conducted during visiting in IE Dept., CUHK, † Project Lead, ‡ Corresponding Author

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Scaling Environments for Agents (SEA).

Figure 1: A chemist controlling an automative synthesis system (on the left of the image, not fully recorded). This image is a typical example of the most frequently used equipment control method via computer I/O such as keyboards and mouse.

configuration tasks daily, such as setting robot motion parameters or adjusting networking device configurations. Significant human workload could be alleviated if these operations were automated and conducted in an unmanned manner, using control scripts generated by equipment management frameworks for system adjustments.

There have been some previous attempts in building specialized industrial software for automatic equipment control, such as typical Programmable Logic Controller (PLC) environments like Unity Pro [1], IndraWorks [2], and TwinCAT [3]. These tools are designed to facilitate automation by leveraging scripting capabilities in manufacturing and plant operations. Yet, as flexible manufacturing continues to advance, these customized industrial software solutions have gradually revealed several shortcomings. Such software often suffers from lengthy development cycles, limited flexibility, and poor generalizability. Each new manufacturing technique typically demands additional software engineering efforts, including customized PLC configurations and the development of new control scripts. These tools, lacking the ability to scale across diverse industrial platforms, are rarely capable of autonomously adapting to new user interfaces or workflows without extensive re-engineering.

Recent investigations in Large Language Models (LLMs) have brought new possibilities in AI-assisted industrial device control. Leveraging the reasoning and generalization abilities of LLMs, prior works have developed LLM-driven coordinators for automating industrial device management [4] across diverse platforms, fine-tuned LLM agents for reliable control script generation [5], and efficient industrial information matching based on Retrieval-Augmented Generation (RAG) [6, 7].

Despite the promising picture of LLM-assisted factory management, a common assumption in previous works [4–7] is the availability of programming-based control interfaces for industrial devices. However, in most factories today, it remains common for industrial computers to be operated via I/O devices, such as keyboards, mice, and touchscreens (see Fig. 1 for example). These Graphical User Interface (GUI)-based configurations rely on skilled workers to manually interact with specialized software GUIs rather than machine-understandable programming interfaces, making LLM-supervised device automation particularly challenging.

The concept of LLM-empowered General Computer Control (GCC), which develops AI agents capable of interacting with arbitrary GUI-based software to perform tasks in a manner similar to human users, could fill the critical gap between the assumption in previous works [4–7] and the practical situation in real factories. However, prior research on LLM-GCC has primarily focused on navigation in everyday scenarios, such as interacting with webpage[8–13], mobile device[14–19], or regular office software[20–24]. These environments are typically structured, homogeneous, and rely on system-level representations like Document Object Model (DOM) trees. Industrial GUI environments, on the other hand, are far more complex. They are often visually diverse, closed-source, and highly domain-specific, featuring dynamic menus, real-time system feedback, and extensive parameter spaces. Additionally, operations in industrial settings frequently have mission-critical consequences, requiring precise execution and high reliability.

While LLM-GCC is both critical and challenging for industrial applications, existing research has made little progress in this domain. To date, there is not even a publicly available dataset specifically addressing LLM-GCC for industrial scenarios, let alone further exploration of LLM pre-training or post-training tailored to these applications.

Building on the above observations, this paper attempts to address the absence of an industrial GCC dataset. Through collaboration with real, operational factories, we collected a real-world, GUI-based dataset that captures workers' interactions while operating industrial equipment. This dataset, known as IndusGCC, is the first publicly available benchmark suite specifically designed for LLM-GCC in industrial settings, representing an important step towards fully automatic factory management.

Major contributions and key insights of this paper are summarized as follows.

Introduction of a Large-Scale Dataset: We present IndusGCC, a large-scale dataset encompassing 448 real-world tasks across seven industrial domains, including robotics control, mission-critical industrial networking, and manufacturing automation. Each task instance is paired with multimodal human interaction data that are synchronized in time, including screen recordings, mouse positions, mouse and keyboard events, and textual task descriptions. This dataset is designed to provide rich supervision signals for GUI-level code generation and facilitate future research in this area.

Comprehensive Evaluation Metrics: We propose a set of evaluation metrics that closely align with real-world usability. Task success is measured based on functional code equivalence – specifically, whether the LLM-generated control script produces the same system behavior as a gold-standard code reference, which represents how a human worker would operate the equipment. Additionally, we assess step completeness of the LLM-generated scripts using the Smith-Waterman algorithm to compare Abstract Syntax Tree (AST) sequences, capturing structural alignment between the LLM-generated code and the reference code.

Benchmarking Mainstream LLMs: Using the novel dataset and evaluation methods, we benchmark several mainstream LLMs to assess their performance in industrial control script generation. The results demonstrate the feasibility of LLM-GCC in industrial contexts while highlighting notable performance gaps. These gaps reveal the need for improvements in spatial perception, temporal understanding, and behavior planning in current models. We believe this benchmarking effort provides a meaningful foundation for future research in this area.

2 Related work

Dataset and Benchmark for LLM-GCC: A growing number of works have emerged in building benchmarks for LLM-GCC. Existing datasets vary significantly in terms of task type and observation space (i.e. data types that is allowed as the agent's input).

Early attempts at constructing benchmarks for LLM-GCC largely focused on web navigation tasks, where well-structured environments like DOM Trees offered deterministic state representations [10, 11]. Over time, research expanded toward more diverse and realistic settings such as mobile device control, aiming to approximate the complexity of human-computer interaction in mobile operating systems [17–19]. In recent datasets, tasks go beyond simple navigation, increasingly encompassing daily-use general applications including document editing, video processing, gaming, and even software development [16, 20–22, 25–28].

To support such complexity, the design of the GCC task's observation space has also undergone significant evolution. Initial benchmarks provided only natural language prompts as input [25, 26, 29], while later works introduced well-structured control representations such as DOM trees and Android Extensible Markup Language (XML) data to bridge the gap between high-level instructions and low-level UI elements [11, 30]. Variants of the "Set-of-Mark" (SOM) format [31] became standard in many recent mobile- and web-based datasets due to their modularity and adaptability. However, these formats inherently rely on the availability of structured access, which limits their applicability to closed-source software environments. With the advent of multimodal LLMs (MLLMs), recent efforts have begun to fuse visual inputs – screenshots or videos – with SOM, as richer observation modalities [9, 12, 13, 18–23, 32–35]. Yet, most of these visual benchmarks continue to serialize perception through SOM-like abstractions rather than embracing fully end-to-end visual modeling.

IndusGCC pushes this frontier by addressing task setups where structured access is entirely absent, which is in fact a common case in closed-source industrial software that exposes neither DOM trees nor accessibility APIs for safety reasons. We adopt a minimal observation setup, where the agent must rely solely on raw screen pixels and natural language task instructions to perceive state and make decisions. This setting reflects a realistic and highly constrained industrial use case, where interface complexity and software opacity preclude structured interaction, and intelligent agents must operate purely from visual and linguistic grounding.

Equally important, high-quality industrial interaction data is exceptionally scarce: it must be captured through direct engagement with operational systems in real production environments. To address this fundamental gap, our dataset collects 448 authentic tasks, recorded across seven key application domains, each grounded in routine yet critical factory operations. These tasks include configuring wireless routers in network management rooms, designing signal flows for radio-frequency testbeds, performing robotic motion planning in robot calibration stations, optimizing manipulator trajectories in assembly-line teaching scenarios, inspecting protocol packets during industrial system diagnostics, programming weld sequences in automated welding workcells, and orchestrating reagent pipelines in chemical laboratories. This makes IndusGCC the first of its kind to support vision-based, instruction-driven control across such a broad and practically grounded set of real-world industrial contexts.

Evaluation for LLM-GCC: In recent studies, evaluation methods of LLM-GCC systems have progressed from early manual annotation schemes – often used in small-scale studies or proof-of-concept datasets – to more systematic and scalable automatic evaluation protocols [14, 15, 17]. Automatic evaluation is now standard across most recent benchmarks and can be broadly categorized into offline, interactive, and hybrid approaches. Offline evaluation compares model outputs to predefined ground truth without requiring execution in a live environment [10, 13, 24, 26]. Interactive evaluation executes model-generated actions within a real or simulated interface and determines success based on the resulting system state. Within interactive settings, some datasets operate in open environments (e.g., browsers or operating systems with uncontrolled variability), while others rely on customized mirrored environments designed to ensure reproducibility and safety [8, 9, 11, 12, 16, 19–23, 27, 29, 34, 35]. Hybrid evaluation combines offline and interactive components – often scoring intermediate reasoning steps or plans offline, while verifying final execution outcomes via interactive runs [11, 18, 25, 28].

In terms of evaluation metrics, task success rate has emerged as the most widely adopted and critical indicator. As a result-driven metric, it reflects the agent's overall perception, planning, and decision-making competence across varying task types. To complement this binary outcome measure, several benchmarks introduce step-wise completion or efficiency scores – quantifying the fraction of gold-standard steps reproduced by the agent – which can reveal omissions or redundancies in generated control sequences [9, 13–17, 25, 30]. Other metrics include action accuracy, such as the validity of mouse click coordinates [18], and intermediate state correctness, which evaluates partial progress toward goals or correct identification of UI elements during execution [10, 15, 18, 26, 28, 30, 32–34].

Compared with prior works, our evaluation protocol introduces two key innovations: First, we employ richer and more fine-grained evaluation metrics beyond traditional task success rates. Each model prediction is represented as executable PyAutoGUI-based control code, enabling multiple dimensions of assessment. We adopt an LLM-as-judge mechanism to evaluate whether predicted and reference scripts are functionally equivalent – based not merely on syntax but on their actual control effect. This also eliminates the need for researchers to manually configure or simulate execution environments. To further evaluate the internal structure of action sequences, we also compute step-level similarity and operation-level precision using a Smith-Waterman alignment algorithm over AST. This evaluation framework provides an interpretable and rigorous measure of outcome quality and procedural fidelity.

Second, IndusGCC addresses a challenge often overlooked in previous datasets: the problem of operational tolerance. In real industrial scenarios, some control actions – especially those involving mouse positioning – do not require pixel-perfect accuracy, but rather fall within tolerable spatial ranges. We annotate these tolerances explicitly IndusGCC, allowing the evaluation process to distinguish between semantically correct and functionally invalid actions. This enables models to better reason about control affordances and prevents unfair penalization of predictions that are practically acceptable but not identically matched. As a result, IndusGCC provides a realistic and robust foundation for measuring model performance under real-world constraints.

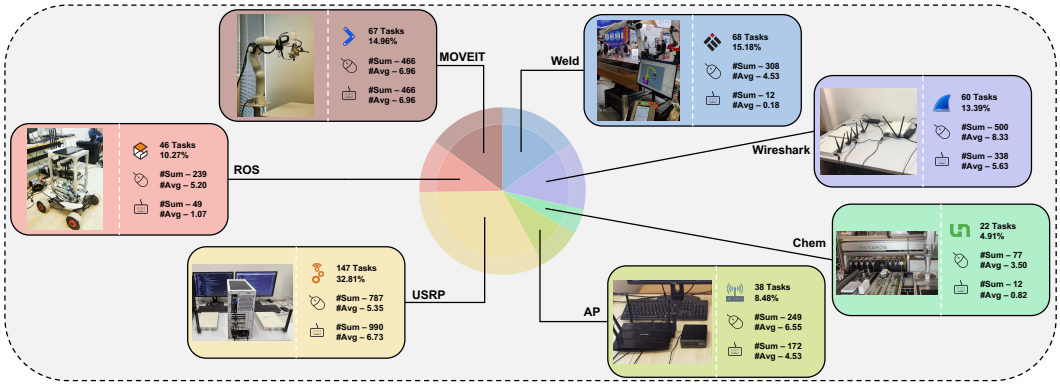

Figure 2: Dataset Overview and Statistics. Each subset presents the associated industrial equipment, industrial control software, and statistical information.

3 Dataset and Baseline

3.1 Dataset Overview

We construct the first large-scale industrial GUI operation dataset by collecting human interaction videos from seven real-world industrial scenarios, each of which reflects practical production needs and high operational complexity. These scenarios cover:

1. Robotic arm control (precise motion parameter tuning and task execution by MOVEIT[36]);
2. Network device configuration (e.g., access point setup, router management);
3. USRP simulation (software-defined radio configuration and signal processing)[37, 38];
4. Chemical synthesis process control (reactor settings, safety monitoring);
5. Industrial welding control (parameter optimization and quality inspection);
6. Path planning (robot motion path generation via ROS[39] on Gazebo[40]);
7. Network traffic analysis (Wireshark-based inspection and diagnostic reporting)[41].

Since IndusGCC is designed to capture the fine-grained visual and behavioral diversity of industrial GUIs, while retaining mission-critical operational details, we recorded the complete process of the human-performed task in a real-world software environment, including mouse movements, clicks, and keyboard inputs. Fig. 2 presents the statistics of IndusGCC, including task quantity proportion and average number of steps per task across the seven domains.

We also demonstrated how we preprocessed the raw data, including data collection, task division and ground truth code generation (see Fig. 3 for an illustration of the whole process). Section 3.2 3.3 and 3.4below introduce the three-step procedure in detail.

3.2 Data Collection

During the data collection process, we found that some industrial equipment provided root access to the system. This allowed us to utilize a custom recording script to capture both system logs and screen recordings during workers' operations. However, other equipment did not grant root access, meaning we were unable to retrieve system logs for analyzing mouse clicks or keyboard inputs. Nevertheless, we were still able to obtain screen recordings of workers' interactions with the machine, and then we manually annotated the mouse-clicking or keyboard-inputting behaviors of the video to maintain the consistency of data structure. It is worth noting that the mouse and keyboard interaction data were collected solely for generating gold-standard control scripts, which serve as benchmarks for evaluating LLM-GCC performance. These data are not used as inputs when IndusGCC is used for model testing. Due to page limit, implementation details of data collection are given in Appendix B.

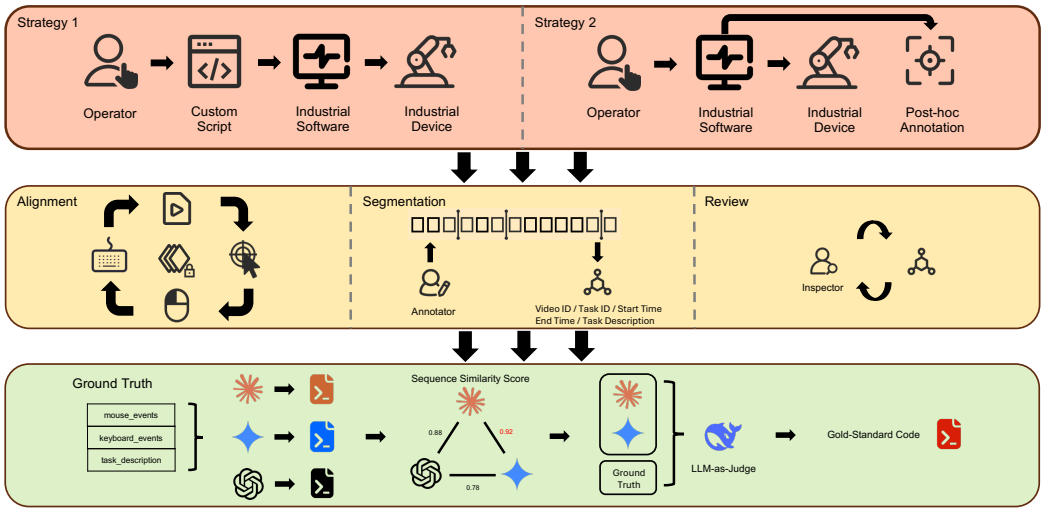

Figure 3: An overview of the three-step procedure introduced in Sections 3.2, 3.3, and 3.4.

3.3 Task Segmentation

Since a single recording session could encompass multiple distinct operational goals, segmenting the raw recordings into well-defined tasks was essential for creating a consistent and reusable dataset. Ideally, each task in our dataset functioned as an independent unit with a clearly defined start point, end point, and operational objective. To achieve this, the construction process of IndusGCC followed a three-phase workflow:

Alignment: All raw recordings underwent a rigorous alignment process. Video frames, mouse events, mouse positions, and keyboard events were synchronized based on frame indices, ensuring a frame-level lock across all modalities. This guaranteed that every task segment contained perfectly matched visual and interaction data, preventing temporal drift between modalities. The alignment process also ensured that when tasks were later segmented, each segment remained internally consistent and ready for step-by-step reproduction.

Segmentation: After recording, each video was assigned to a member of the annotation team who was familiar with the equipment operations depicted in the video. The annotator segmented the video into multiple tasks according to 1) a unified set of guidelines to ensure dataset-wide consistency, and 2) their expertise in the specific equipment's operation. For each identified task, the annotator created a structured task description that included 1) the video ID, 2) the task ID, 3) the start and end timestamps, and 4) the task description. The task description consisted of two key elements: the overall task goal and a step-by-step, high-level description of the control interactions within the task.

Review: The segmented tasks and their detailed descriptions were reviewed by another team member with expertise in the relevant machine operations. This review ensured that: 1) task boundaries were technically valid, 2) execution steps were complete and accurate, and 3) task semantics aligned with real industrial workflows.

3.4 Ground Truth Code Generation

To establish a reliable code reference for model evaluation, this paper, positioned as a dataset and benchmark effort, adopts a multi-agent collaboration framework to generate ground truth for each operation video on industrial equipment. The framework consists of two phases: Cooperative Code Generation and LLM as a Judge.

In the Cooperative Code Generation phase, three LLMs – GPT-4o, Gemini-2.5-Pro, and Claude 4 Sonnet – were independently tasked with generating executable control script based on the following inputs: mouse positions, mouse events, keyboard events, and task descriptions. Each model produced a full code script intended to replicate the recorded user operation.

After independent code generation at each LLM, we then analyze the similarities between resulting codes to find the most consistent pair of codes. This is motivated by our observation that if two LLMs reach agreement on the operation sequence (i.e., mouse clicking and keyboard input), then the resulting code is very likely to be the correct one. To identify the most consistent pair of candidate solutions, pairwise code similarity scores were computed for all possible model pairs. The pair with the highest score was selected as the primary reference set for the next phase.

For the computation of pairwise code similarity, we employ the Smith-Waterman local alignment algorithm [42] as in previous works [43, 44]. The algorithm measures the structural correspondence between operation sequences by identifying optimally matching subsequences rather than enforcing a global alignment. This ensures that partial but functionally critical overlaps in operation paths are preserved in the scoring process [43].

Specifically, let the operating sequence of one control script be $A_T = (a_1^T, a_2^T, \ldots, a_m^T)$ and the other be $A_S = (a_1^S, a_2^S, \ldots, a_n^S)$, in which each element within the sequence represents a mouse clicking operation or a keyboard inputting operation. The Smith-Waterman algorithm constructs a dynamic programming matrix H where $H_{i,j}$ represents the highest local alignment score between the prefixes A_T and A_S. The recurrence relation is given by:

$$H_{i,j} = max \begin{cases} 0, \\ H_{i-1,j-1} + score(a_i^T, a_j^S), \\ H_{i-1,j} - \delta, \\ H_{i,j-1} - \delta \end{cases} \tag{1}$$

where$score(a_i^T, a_j^S)$ is the substitution score:

$$score(a_i^T, a_j^S) = \begin{cases} 2, a_i^T = a_j^S \\ -1, a_i^T \neq a_j^S \end{cases} \tag{2}$$

and δ is the gap penalty, which $\delta = -1$.
The algorithm initializes $H_{i,0} = H_{0,j} = 0$ and the similarity score is taken as:

$$SimilarityScore\left(A_T, A_S\right) = \frac{max_{(i,j)} H_{i,j}}{n * score\left(a_i^T, a_j^S\right)|_{a_i^T \neq a_j^S}} \tag{3}$$

In our context, A_T and A_S correspond to sequences of atomic operations extracted from the execution traces. The final Smith–Waterman score is normalized by the length of the reference sequence to ensure comparability across tasks.

In the second phase, LLM as a Judge, a fourth high-performing model, DeepSeek-V3, was introduced. DeepSeek-V3 was provided with all inputs available to the first-stage models, along with the two most similar code sequences from the previous phase and their similarity comparison results. Using this comprehensive context, DeepSeek-V3 synthesized a refined code sequence that combined the strengths of both candidates while resolving any inconsistencies.

This two-phase LLM collaboration framework aims to minimize potential biases associated with any single model. The resulting code sequences were then reviewed by human experts to ensure correctness and operational equivalence within the annotated clickable regions. The validated outputs were ultimately published as the gold-standard reference code for the corresponding task segments.

4 Evaluation

4.1 Evaluation Metrics

We compare the machine control script generated by the tested model (TestCode) with the gold-standard code produced through the pipeline described in Section 3.4 (SampleCode). Evaluation is performed across four complementary dimensions designed to capture both semantic correctness and structural fidelity: 1) Task Success Rate, 2) Sequence Similarity Score, 3) Operation Hit Rate, and 4) Operation Redundancy Rate.

Task Success Rate: The first dimension measures whether TestCode and SampleCode are functionally equivalent in accomplishing the intended task. To evaluate this, we employ GPT-4o as an automatic judge: it receives both scripts together with the task description and determines whether they can

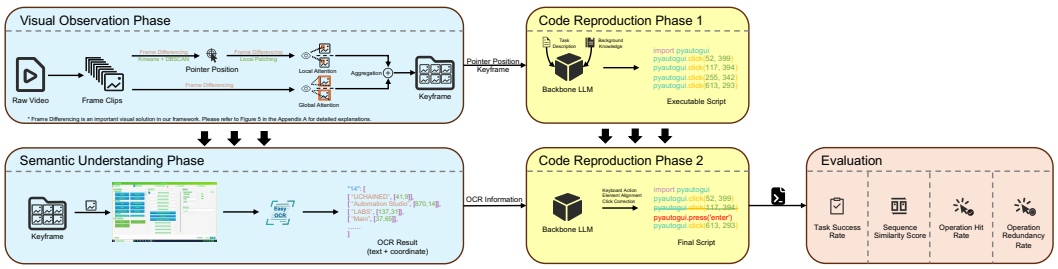

Figure 4: An illustration of the evaluation framework.

be regarded as semantically equivalent. This metric reflects end-to-end task success, focusing on functional correctness rather than surface-level similarity.

Sequence Similarity Score: The second dimension evaluates how closely the structural execution flow of TestCode matches that of SampleCode. We compute this by aligning the two operation sequences using a local sequence alignment algorithm. Intuitively, the algorithm rewards matching operations, penalizes mismatches and gaps, and ultimately identifies the best-matching subsequences. The final score reflects the degree of overlap in execution paths, normalized by task length to allow fair comparison across tasks.

Operation Hit Rate: The third dimension focuses on parameter-level accuracy for critical GUI operations. We extract mouse-related events (e.g., *click(), moveTo(), dragTo()*) and keyboard-related events (e.g., *typewrite(), press(), keyDown()*). A mouse event in TestCode is counted as correct if its coordinates fall within the annotated tolerance region of the corresponding SampleCode event. For keyboard events, only exact string matches are accepted. The hit rate is then calculated as the proportion of correctly executed operations relative to all required operations, thereby quantifying precision in reproducing human interactions.

Operation Redundancy Rate: Finally, we assess whether the generated code contains unnecessary steps beyond those required for successful execution. Redundancy is defined as the proportion of extraneous operations in TestCode relative to the number of valid steps. This rate can be either positive, indicating that TestCode introduces redundant operations compared to the gold-standard script, or negative, suggesting that TestCode omits essential steps. Values closer to zero imply minimal redundancy, reflecting a more faithful and efficient reproduction of the intended procedure.

4.2 Evaluation Framework

To realize fully automatic control script generation based on video inputs, specialized tools have been developed to better understand the video input [6]. These tools can precisely identify key operation frames from the given raw video and conduct optical character recognition (OCR) for each identified frame to extract operation information, significantly reducing the workload of later LLM analysis. While the ultimate goal of LLM-GCC is to develop powerful AIs that are able to address the video-code transformation with a single model, this paper, as a dataset and benchmarking effort, validates the feasibility of this line of research with the cooperation of multiple models. Specifically, our later experiments leveraged the tool open-sourced in [6] as pre-processing (see those highlighted in blue in Fig. 4), and we then used text-based models to generate executable Python scripts restricted to the PyAutoGUI API [45] for GUI control (see those highlighted in yellow in Fig. 4). This was followed by an evaluation stage that considered all evaluation metrics described in Section 4.1 (see those highlighted in red in Fig. 4). Due to space constraints, we do not present further explanation of the evaluation framework. We refer interested readers to Appendix A for details.

5 Experiment and Discussion

Experimental Results: The evaluation framework introduced above enables text-based LLMs to interpret GUI interfaces and generate executable operation paths without native visual capabilities. We benchmarked eight mainstream text-based models across diverse industrial GUI scenarios. Experimental results indicate that none of these models can serve as a plug-and-play solution for perfectly operating industrial GUI systems. Even the best-performing model, GPT-4 on the Chem

Table 1: Experiment results of Task Success Rate, Sequence Similarity Score,Operation Hit Rate and Operation Redundancy Rate on mainstream models

Metrics	Model	Data Subset						
		AP	Chem	MOVEIT	ROS	USRP	Weld	Wireshark
Task Success Rate	claude-sonnet-4-20250514	**7.90%**	18.18%	**14.93%**	6.52%	4.76%	1.47%	0.00%
	llama3.1-405b	5.26%	9.09%	7.58%	0.00%	18.75%	1.47%	0.00%
	gpt-4o	2.63%	13.64%	8.96%	2.17%	**19.18%**	2.94%	1.67%
	grok-3	5.26%	18.18%	4.48%	**8.70%**	5.44%	**4.41%**	0.00%
	deepseek-v3-250324	0.00%	13.64%	5.97%	0.00%	7.59%	2.94%	1.67%
	doubao-1.5-pro-256k	0.00%	0.00%	5.97%	0.00%	5.48%	2.94%	**3.33%**
	gpt-4	2.63%	**22.73%**	5.97%	2.17%	17.81%	1.47%	**3.33%**
	kimi-k2-instruct	2.63%	9.09%	7.46%	4.35%	8.90%	2.94%	1.67%
Sequence Similarity Score	claude-sonnet-4-20250514	**61.50%**	**73.42%**	**63.99%**	60.35%	**38.31%**	**84.13%**	48.78%
	llama3.1-405b	33.63%	54.33%	40.32%	48.07%	29.05%	56.52%	42.96%
	gpt-4o	35.91%	59.05%	40.92%	50.99%	27.18%	59.29%	35.27%
	grok-3	35.41%	63.59%	46.33%	48.19%	28.20%	63.83%	39.73%
	deepseek-v3-250324	56.94%	70.35%	60.38%	**69.30%**	35.62%	80.80%	**52.88%**
	doubao-1.5-pro-256k	40.52%	52.80%	47.42%	44.69%	25.63%	62.16%	43.10%
	gpt-4	34.94%	56.87%	41.54%	50.12%	26.59%	57.80%	36.30%
	kimi-k2-instruct	56.15%	64.77%	48.23%	58.55%	29.84%	78.11%	48.77%
Operation Hit Rate	claude-sonnet-4-20250514	**65.65%**	58.32%	79.10%	66.01%	45.69%	44.95%	39.33%
	llama3.1-405b	61.70%	54.80%	74.24%	44.93%	49.94%	48.18%	44.73%
	gpt-4o	58.63%	60.59%	83.58%	55.80%	**52.11%**	**51.12%**	47.13%
	grok-3	65.01%	58.89%	**86.57%**	**73.84%**	49.61%	48.38%	**49.71%**
	deepseek-v3-250324	59.55%	**63.43%**	83.58%	62.71%	46.81%	40.76%	41.13%
	doubao-1.5-pro-256k	58.15%	36.48%	58.21%	38.41%	38.85%	33.97%	34.55%
	gpt-4	59.07%	61.16%	85.32%	54.27%	51.25%	46.94%	43.70%
	kimi-k2-instruct	61.70%	57.07%	82.34%	67.32%	44.21%	49.36%	41.20%
Operation Redundancy Rate	claude-sonnet-4-20250514	**0.63%**	24.85%	8.16%	**7.34%**	-26.80%	49.03%	29.36%
	llama3.1-405b	20.54%	28.82%	9.75%	45.22%	28.55%	44.50%	29.85%
	gpt-4o	14.83%	16.44%	-14.59%	29.38%	16.98%	41.64%	**2.18%**
	grok-3	20.80%	20.56%	21.97%	22.61%	23.79%	50.28%	18.15%
	deepseek-v3-250324	17.69%	**9.81%**	8.07%	16.00%	24.65%	46.69%	38.41%
	doubao-1.5-pro-256k	26.53%	26.25%	18.93%	31.48%	19.37%	**33.69%**	11.11%
	gpt-4	18.46%	13.26%	-7.06%	36.81%	**14.42%**	50.54%	20.09%
	kimi-k2-instruct	15.88%	17.79%	**6.87%**	7.53%	19.33%	36.20%	37.06%

subset, achieves only 22.73% task coverage, highlighting the considerable gap between current LLM capabilities and the requirements of real-world industrial applications.

On sequence-level structural fidelity, we observed a more uniform trend: claude-sonnet-4 clearly dominated, achieving the highest similarity scores in nearly all subsets. This indicates that claude-sonnet-4 succeeds in both producing functionally correct scripts and closely reproducing the canonical sequence of human operations. The alignment highlights the model's capacity to capture task-specific procedural structures rather than merely generating semantically equivalent but divergent solutions.

A particularly interesting pattern emerges when comparing fine-grained operation accuracy with redundancy. Across most subsets, higher hit rates correlate with increased redundancy, suggesting that models often insert additional operations as a "safety margin" to maximize coverage of effective interaction zones, even though introduce inefficiency and potential error propagation. Notably, claude-sonnet-4 on the AP subset and deepseek-v3 on the Chem subset stand out as rare cases where high accuracy is achieved without inflating redundancy. These results point to promising directions for balancing correctness with efficiency in future model design.

Observations and Discussions: From an algorithmic perspective, the suboptimal performance of the current framework mainly stems from its limited ability to interpret fine-grained visual information in GUI environments. The reliance on a zero-shot frame-differencing method restricts context adaptation, leading to imprecise recognition of pointer positions, icon shapes, subtle color changes, or menu states. A promising solution is to fine-tune a tool-oriented vision encoder trained on GUI screenshots or adopt multimodal large models that jointly process textual prompts and visual signals, improving alignment between perceived states and operational actions. Moving from zero-shot to

few-shot or domain-adapted fine-tuning may also help internalize common interaction patterns and enhance task success rates.

From a dataset perspective, the current benchmark covers a constrained range of GUI operations, limiting generalization to diverse scenarios. Expanding it with richer manipulations – such as viewport rotation, zooming, scrolling, drag-and-drop, and shortcut key combinations – would create a more representative evaluation space and foster the development of models capable of handling complex, realistic human–computer interaction tasks.

6 Conclusion

This work presents the first general computer control (GCC) dataset for industrial scenarios, covering diverse applications, such as robotics control, mission-critical industrial networking, and automatic chemical synthesis. We design automated data collection pipelines coupled with professional verification to ensure both scalability and quality. Building upon this dataset, we introduce a comprehensive evaluation framework that defines four complementary metrics, i.e., LLM-judged success rate, AST similarity, operation hit rate, and redundancy rate, to enable fine-grained, end-to-end assessment of model performance on industrial GCC tasks. Our benchmarking results across mainstream models reveal their promising capabilities and current limitations, highlighting persistent challenges in accurate operation sequencing and efficiency. This work serves as a preliminary effort in building the domain-specific dataset and calls for further investigation building upon the proposed benchmarks.

7 Acknowledge

The work described in this paper was partially supported by the Research Grants Council of the Hong Kong SAR, China (Grant No. T45-401/22-N), by the 2024 Shenzhen-Hong Kong-Macao Science and Technology Program, Category C (Grant No. SGDX20230821094359004), by the National Natural Science Foundation of China (Grant No. 22504120), by the Fujian Provincial Natural Science Foundation of China (Grant No. 2025J08309).

The authors acknowledge ZJU-Hangzhou Global Scientific and Technological Innovation Center and Shenzhen iManifold Robotics for their support in real-world data records. The authors also acknowledge Prof. Henry Chen from IE department, CUHK, for his support in necessary equipment used in data collection.

References

[1] Schneider Electric. UnityPro, 2019. https://www.se.com/us/en/download/document/UnityPro_EN/[Accessed: May 2025].

[2] Bosch Rexroth. IndraWorks, 2019. https://www.boschrexroth.com/en/us/products/industrial-solutions/electric-drives-and-controls/software-tools/[Accessed: May 2025].

[3] Beckhoff Automation. TwinCAT, 2011. https://www.beckhoff.com/en-en/products/automation/twincat/texxxx-twincat-3-engineering/[Accessed: May 2025].

[4] Hongwei Cui, Yuyang Du, Qun Yang, Yulin Shao, and Soung Chang Liew. Llmind: Orchestrating ai and iot with llm for complex task execution. *IEEE Communications Magazine*, 63(4):214–220, 2025.

[5] Yuyang Du, Qun Yang, Liujianfu Wang, Jingqi Lin, Hongwei Cui, and Soung Chang Liew. Llmind 2.0: Distributed iot automation with natural language m2m communication and lightweight llm agents, 2025.

[6] Kexin Chen, Jiamin Lu, Junyou Li, Xiaoran Yang, Yuyang Du, Kunyi Wang, Qiannuan Shi, Jiahui Yu, Lanqing Li, Jiezhong Qiu, Jianzhang Pan, Yi Huang, Qun Fang, Pheng Ann Heng, and Guangyong Chen. Chemist-x: Large language model-empowered agent for reaction condition recommendation in chemical synthesis, 2025.

[7] Kexin Chen, Yuyang Du, Junyou Li, Hanqun Cao, Menghao Guo, Xilin Dang, Lanqing Li, Jiezhong Qiu, Pheng Ann Heng, and Guangyong Chen. Chemminer: A large language model agent system for chemical literature data mining, 2025.

[8] Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. Autowebglm: A large language model-based web navigating agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 5295–5306, New York, NY, USA, 2024. Association for Computing Machinery.

[9] Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, and Zhengyang Wu. Webcanvas: Benchmarking web agents in online environments, 2024.

[10] Xing Han Lù, Zdeněk Kasner, and Siva Reddy. Weblinx: real-world website navigation with multi-turn dialogue. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[11] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 20744–20757. Curran Associates, Inc., 2022.

[12] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6864–6890, Bangkok, Thailand, 08 2024. Association for Computational Linguistics.

[13] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: towards a generalist agent for the web. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

[14] Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *CoRR*, abs/2401.16158, 2024.

[15] Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 2686–2710. Curran Associates, Inc., 2024.

[16] Jingxuan Chen, Derek Yuen, Bin Xie, Yuhao Yang, Gongwei Chen, Zhihao Wu, Li Yixing, Xurui Zhou, Weiwen Liu, Shuai Wang, Kaiwen Zhou, Rui Shao, Liqiang Nie, Yasheng Wang, Jianye Hao, Jun Wang, and Kun Shao. Spa-bench: A comprehensive benchmark for smartphone agent evaluation, 2025.

[17] Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Androidinthewild: A large-scale dataset for android device control. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 59708–59728. Curran Associates, Inc., 2023.

[18] Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. AndroidLab: Training and systematic benchmarking of android autonomous agents. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2144–2166, Vienna, Austria, July 2025. Association for Computational Linguistics.

[19] Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents, 2025.

[20] Pei Yang, Hai Ci, and Mike Zheng Shou. macosworld: A multilingual interactive benchmark for gui agents, 2025.

[21] Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Jang, and Zack Hui. Windows agent arena: Evaluating multi-modal os agents at scale, 2024.

[22] Henry Hengyuan Zhao, Kaiming Yang, Wendi Yu, Difei Gao, and Mike Zheng Shou. Worldgui: An interactive benchmark for desktop gui automation from any starting point, 2025.

[23] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 52040–52094. Curran Associates, Inc., 2024.

[24] Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem AlShikh, and Ruslan Salakhutdinov. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol, editors, *Computer Vision – ECCV 2024*, pages 161–178, Cham, 2025. Springer Nature Switzerland.

[25] Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: an analytical evaluation board of multi-turn llm agents. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2025. Curran Associates Inc.

[26] Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants, 2023.

[27] Ruisheng Cao, Fangyu Lei, Haoyuan Wu, Jixuan Chen, Yeqiao Fu, Hongcheng Gao, Xinzhuang Xiong, Hanchong Zhang, Yuchen Mao, Wenjing Hu, Tianbao Xie, Hongshen Xu, Danyang Zhang, Sida Wang, Ruoxi Sun, Pengcheng Yin, Caiming Xiong, Ansong Ni, Qian Liu, Victor

Zhong, Lu Chen, Kai Yu, and Tao Yu. Spider2-v: how far are multimodal agents from automating data science and engineering workflows? In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2025. Curran Associates Inc.

[28] Lawrence Jang, Yinheng Li, Dan Zhao, Charles Ding, Justin Lin, Paul Pu Liang, Rogerio Bonatti, and Kazuhito Koishida. Videowebarena: Evaluating long context multimodal agents with video understanding web tasks, 2025.

[29] Léo Boisvert, Megh Thakkar, Maxime Gasse, Massimo Caccia, Thibault Le Sellier De Chezelles, Quentin Cappart, Nicolas Chapados, Alexandre Lacoste, and Alexandre Drouin. Workarena++: Towards compositional planning and reasoning-based common knowledge work tasks. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 5996–6051. Curran Associates, Inc., 2024.

[30] Qi Chen, Dileepa Pitawela, Chongyang Zhao, Gengze Zhou, Hsiang-Ting Chen, and Qi Wu. Webvln: vision-and-language navigation on websites. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024.

[31] Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v, 2023.

[32] Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. AssistantBench: Can web agents solve realistic and time-consuming tasks? In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8938–8968, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[33] Shulin Tian, Ziniu Zhang, Liangyu Chen, and Ziwei Liu. Mmina: Benchmarking multihop multimodal internet agents, 2025.

[34] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[35] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024.

[36] Michael Görner, Robert Haschke, Helge Ritter, and Jianwei Zhang. Moveit! task constructor for task-level motion planning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 190–196, 2019.

[37] GNU Radio. GNU Radio, 2023. `https://www.gnuradio.org/`[Accessed: May 2025].

[38] Ettus Research. USRP Hardware Driver, 2018. `https://github.com/EttusResearch/uhd`[Accessed: May 2025].

[39] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

[40] Open Robotics Foundation. Gazebo, 2024. `https://gazebosim.org/`[Accessed: May 2025].

[41] Wireshark Foundation. Wireshark, 2024. `https://www.wireshark.org/`[Accessed: May 2025].

[42] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[43] Yanming Yang, Zhilei Ren, Xin Chen, and He Jiang. Structural function based code clone detection using a new hybrid technique. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 01, pages 286–291, 2018.

[44] Shahad S. Omer and Asma'a Y. Hammo. A comparative study for language independent code clone detection. *AIP Conference Proceedings*, 3211(1):030050, 05 2025.

[45] Al Sweigart. PyAutoGUI, 2019. `https://github.com/asweigart/pyautogui`[Accessed: May 2025].

[46] A. Dutta, A. Gupta, and A. Zissermann. VGG image annotator (VIA). http://www.robots.ox.ac.uk/ vgg/software/via/, 2016. Version: X.Y.Z, Accessed: INSERT_DATE_HERE.

[47] Abhishek Dutta and Andrew Zisserman. The VIA annotation software for images, audio and video. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, New York, NY, USA, 2019. ACM.

A Algorithm details of Evaluation Framework

Visual Observation Phase In GUI-based industrial software operations, the mouse pointer or the input cursor serves as the primary locus of human visual attention. We define key events as either mouse clicks or keyboard inputs, and the corresponding video frames in which these events occur as key frames.

We found that most key events on industrial GUIs exhibit "trigger-and-feedback" patterns—either local changes (e.g., button highlighting, text field activation) or global changes (e.g., page refresh or redrawing) upon interaction. By measuring both local and global pixel-level differences between consecutive frames, the framework can detect significant state transitions and pinpoint the mouse pointer location at event time. This hybrid approach is inspired by human selective attention, and we term it the Global–Local Attention.

To automatically endow the model with the Global–Local Attention capability to achieve automatic detection of mouse pointers and keyframes, we employ a Frame Differencing method, which computes the absolute difference of each pixel between two consecutive frames to quantify the degree of visual change. As illustrated in Figure 5, Frame 2 introduces a new popup window compared with Frame 1. By rendering the result, the non-overlapping regions become visible, clearly highlighting the new elements (in this case, located inside the red box on Frame 2).

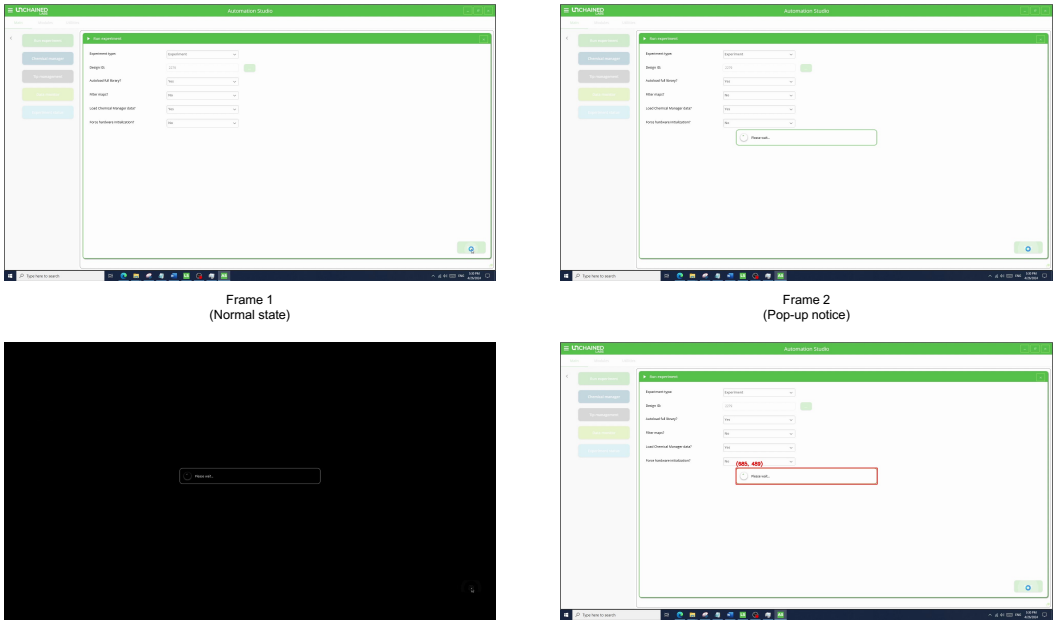

Figure 5: An example demonstrating Frame Differencing method

Semantic Understanding Phase While modern LLMs possess strong language understanding capabilities, their performance in spatial grounding remains limited. To bridge this gap, we integrate Optical Character Recognition (OCR) into the workflow. Using the EasyOCR library, we extract textual content from each key frame, along with the corresponding bounding-box coordinates. This enables a joint semantic–spatial representation of the GUI state, facilitating accurate mapping between textual UI elements and their screen positions in downstream reasoning steps.

Code Reproduction Phase Code generation proceeds in a two-stage chat with the LLM backbone:

Initial Code Drafting — The model is provided with the dataset background, task goal description, and a sequence of detected key frames along with predicted mouse click coordinates. The model is instructed to output a first-pass executable script using only pyautogui functions for GUI manipulation.

Code Refinement — The model receives the OCR-derived semantic–spatial annotations of key frames and is tasked with refining the initial script. This may include inserting missing keyboard actions, correcting mouse click coordinates and aligning interactions with text-labeled interface elements.

The final output of this two-stage generation process is an executable Python program, which is subsequently compared against the gold-standard SampleCode using the evaluation metrics from Section 4.1.

B Implementation details of Data Collection

We now detail our data collection process for the two scenarios mentioned in Section 3.2 as follows.

Strategy One: Script-Based Recording. For equipment with root access, we used the script open-sourced alongside the dataset at https://github.com/Golden-Arc/IndusGCC. Once initiated via a terminal, the script continuously captured:

1) *Full-Screen Video*: Recorded at a resolution of 1920×1080 and a frame rate of 60 FPS, utilizing FFmpeg for efficient real-time encoding.

2) *Mouse Movement Trajectory*: Captured as tuples of (timestamp, x-coordinate, y-coordinate) for each frame, ensuring sub-frame temporal resolution.

3) *Mouse Events*: Logged with timestamp, x-coordinate, y-coordinate, button type (e.g., left/right), and button state (e.g., pressed/released).

4) *Keyboard Events*: Recorded with timestamp, key identifier, and key states (e.g., pressed/released).

The script is designed for stable operation on commonly used operating systems where equipment control software is deployed (such as the latest Ubuntu Release and Windows 10/11) to ensure cross-platform reliability for the LLM-GCC agent. Operators must manually start the script from a command-line terminal and terminate it by entering an interrupt command.

To ensure consistent and high-quality data collection, each equipment operator underwent a structured training session and followed a step-by-step tutorial prior to recording. Operators conducted three preliminary trial runs to confirm environment compatibility, validate accurate script output, and ensure adherence to dataset specifications. During the formal recording phase, operators were not provided with pre-defined task scripts; instead, they were encouraged to perform operations that reflect real-world industrial production tasks, particularly those involving machine configuration. To maintain data clarity and reproducibility, operators were instructed to minimize errors and redundant actions, ensuring that recorded sequences are concise, executable, and representative of practical use cases.

Strategy Two: Post-Hoc Manual Annotation. In certain industrial environments, running the recording script was not feasible due to restricted system access. In such cases, we resorted to manual post-hoc annotation based on the screen recording videos. Specifically, each video was first processed at a standard frame rate of 10 FPS and a resolution of 1920×1080. The videos were then decomposed into continuous frames using OpenCV, and each frame was annotated with the VGG Image Annotator (VIA), a lightweight, open-source annotation tool developed by the Visual Geometry Group [46, 47]. To ensure consistency across data acquisition modes, the annotation schema for manual labeling mirrored that of the automated recording script. Annotators were instructed to label the following information for each frame: 1) mouse position coordinates, including timestamp and the location of the mouse; 2) mouse click events, including button type and mouse state; 3) keyboard events, including timestamp, key identifier, and the state of the keyboard.