GEM: Gaussian Embedding Modeling for Out-of-Distribution Detection in GUI Agents

Zheng Wu **Pengzhou Cheng** **Zongru Wu** **Lingzhong Dong** **Zhuosheng Zhang**[*]
School of Computer Science, Shanghai Jiao Tong University
Email: {wzh815918208, zhangzs}@sjtu.edu.cn

Abstract

Graphical user interface (GUI) agents have recently emerged as an intriguing paradigm for human-computer interaction, capable of automatically executing user instructions to operate intelligent terminal devices. However, when encountering out-of-distribution (OOD) instructions that violate environmental constraints or exceed the current capabilities of agents, GUI agents may suffer task breakdowns or even pose security threats. Therefore, effective OOD detection for GUI agents is essential. Traditional OOD detection methods perform suboptimally in this domain due to the complex embedding space and evolving GUI environments. In this work, we observe that the in-distribution input semantic space of GUI agents exhibits a clustering pattern with respect to the distance from the centroid. Based on the finding, we propose GEM, a novel method based on fitting a Gaussian mixture model over input embedding distances extracted from the GUI Agent that reflect its capability boundary. Evaluated on eight datasets spanning smartphones, computers, and web browsers, our method achieves an average accuracy improvement of 23.70% over the best-performing baseline. Analysis verifies the generalization ability of our method through experiments on nine different backbones. The codes are available at `https://github.com/Wuzheng02/GEM-OODforGUIagents`.

1 Introduction

Recently, graphical user interface (GUI) agents [1, 2, 3, 4, 5] have emerged as an intriguing paradigm to human-computer interaction, capable of autonomously executing user instructions and performing human-like control on intelligent terminal devices such as smartphones, computers, and web browsers. The common approach to building GUI agents involves post-training multimodal large language models (MLLMs) using high-quality trajectory data to enhance key task capabilities such as perception [6, 7], reasoning [8, 9, 10], and reflection [11, 12].

Despite notable advances in instruction following, GUI agents remain vulnerable to out-of-distribution (OOD) risks—executing instructions that violate environmental constraints (e.g., non-existent functions or applications) or exceed the agent's current capabilities. These failures can lead to task breakdowns or even pose security threats. In real-world applications, OOD risks for GUI agents come with two primary forms: (i) **Internalization-OOD**, where the agent operates in domain-specific environments (e.g., a particular type of smartphones or vehicle cabins) but incorrectly assumes the presence of unsupported capabilities; (ii) **Extrapolation-OOD**, where the agent encounters instructions tied to dynamic or evolving environments (e.g., new third-party applications).

As illustrated in Figure 1, following OOD instructions and executing an incorrect action path—such as unintentionally resetting a smartphone—can result in critical failures. Therefore, it is essential for

[*]Corresponding author.

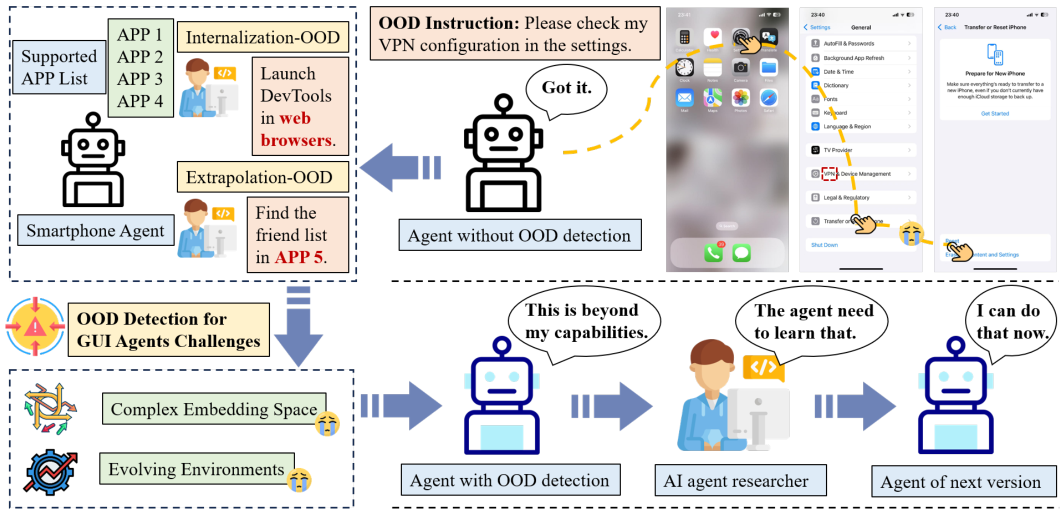
Supported APP List
APP 1
APP 2
APP 3
APP 4
Internalization-OOD
Launch DevTools in web browsers.
Extrapolation-OOD
Find the friend list in APP 5.
Smartphone Agent
OOD Instruction: Please check my VPN configuration in the settings.
Got it.
Agent without OOD detection
OOD Detection for GUI Agents Challenges
Complex Embedding Space
Evolving Environments
This is beyond my capabilities.
The agent need to learn that.
I can do that now.
Agent with OOD detection
AI agent researcher
Agent of next version

Figure 1: Comparison between an agent w/ and w/o OOD detection when facing an OOD instruction. Also illustrated are the OOD scenarios and the challenges of OOD detection for GUI agents.

GUI agents to incorporate OOD detection mechanisms that identify tasks beyond their supported scope. This not only mitigates potential operational risks but also enables targeted enhancements to agent capabilities through continued research and development.

Popular OOD detection methods can be broadly categorized [13, 14] into two types: embedding-based approaches [15, 16] and model uncertainty-based approaches [17, 18]. These methods have demonstrated effectiveness in traditional (M)LLM tasks such as summarization [19], visual question answering (VQA) [20], mathematical reasoning [21], and text generation [22]. However, as we will show in Section 3, directly applying these existing OOD detection techniques to the GUI agent domain results in suboptimal performance. Notably, even with the classification threshold calibrated to correctly identify 95% of in-distribution (ID) tasks, the best-performing method still misclassifies between 31.75% and 48.92% of OOD tasks as ID.

Compared with traditional (M)LLM-based OOD detection tasks, OOD detection for GUI agents presents two critical challenges:

• **Complex Embedding Space**: The inputs to GUI agents are inherently complex [23]. Concretely, GUI screens typically contain densely populated UI components [3], leading to much higher information density than traditional MLLM tasks. Besides, the diversity of user instructions further complicates the inference of user intent. This complexity substantially increases the difficulty of effective OOD detection for GUI agents.

• **Evolving Environments**: GUI environments frequently change due to system upgrades or the installation of new third-party applications. As a result, GUI agents must continually adapt their capabilities [24, 25, 26]. This evolving nature complicates OOD detection by demanding both accurate capability assessment and temporal adaptability. Consequently, reliance on static external classifiers becomes increasingly impractical.

To address the challenges above, we propose **GEM**, a novel method based on fitting a Gaussian mixture model (GMM) [27] over input embedding distances extracted from the GUI Agent that reflect its capability boundary. Specifically, we first extract the input embeddings from the encoder layer of the GUI agent on its training data, and fit them into a high-dimensional hypersphere. We then compute the L2-norm distances of all embeddings relative to the centroid of the hypersphere. Under bayesian information criterion (BIC) [28] supervision, we fit a GMM to these distances and define the OOD detection boundary as a configurable number of standard deviations away from each GMM cluster center. Experiments across eight GUI agent datasets spanning smartphones, computers, and web browsers platforms show that our method consistently outperforms all traditional OOD detection methods.

In summary, we make three key contributions:

(i) We present the first systematic analysis of OOD detection for GUI agents and compare various widely adopted OOD detection methods in this domain.

(ii) We propose a novel GMM-based approach that leverages input embedding distances from MLLMs for OOD detection. Our method achieves an average accuracy improvement of 23.70% over the best performing baseline across eight datasets spanning three platform types, consistently outperforming all baseline methods.

(iii) We identify the unique characteristics of OOD detection for GUI agents and provide a comprehensive comparison between our approach and traditional methods, offering new insights into this emerging research area.

2 Related Work

GUI Agents With the rapid advancement of MLLMs [29, 30, 31], numerous cross-platform GUI agent foundation models [32, 33, 34, 35] have emerged, capable of automatically executing user instructions across smartphones, computers, and web browsers. For the technical framework, prompt-based approaches [36, 25, 24] have been developed, leveraging closed-source models to construct agent systems that fulfill user instructions. Additionally, researchers have also explored pre-training [23, 37], supervised fine-tuning (SFT) [38, 39, 40], and reinforcement learning (RL) [41, 42, 43, 44, 45, 46] techniques to further enhance the ability of GUI agents to autonomously complete user instructions. Regardless of their advancements, GUI agents inevitably face capability limitations. In complex and dynamic real-world environments [47, 48, 49], GUI agents are prone to encountering OOD situations. As a result, robust OOD detection mechanisms for GUI agents remain a critical and valuable area of research.

OOD Detection in MLLMs For OOD detection in MLLMs, it is necessary to jointly consider both visual and textual modalities, in contrast to traditional computer vision OOD detection, which only requires modeling the visual modality [50, 51], or LLMs OOD detection, which focuses on textual modality [21, 19]. In the context of MLLMs, some researchers have proposed using maximum concept matching [52] to characterize OOD uncertainty, while others have approached the problem by training models on ID datasets and classifying whether an input image belongs to an unknown category [53]. However, OOD detection for MLLMs remains a challenging research area [54, 55]. Furthermore, for MLLM-based GUI agents, task scenarios are significantly more complex and dynamic [5] compared to traditional MLLM tasks, making OOD detection even more difficult.

3 Investigating the Challenge of OOD Dection for GUI Agents

In this section, we conduct pilot experiments to evaluate the effectiveness of popular OOD detection methods for GUI agents and understand the challenges. We will first provide the problem formulation, and then present the experimental settings followed by the key results and analysis of the pilot study.

3.1 Problem Formulation for OOD Detection for GUI Agents

A GUI agent $\mathcal{F}$ is initially trained on an ID dataset $\mathcal{D}_{\text{ID}} = \{(s_i, x_i)\}_{i=1}^{k}$, consisting of k pairs of screenshots s_i and user instructions x_i. The GUI agent $\mathcal{F}$ learns operational knowledge of operating systems through methods such as SFT and RL on $\mathcal{D}_{\text{ID}}$.

After deployment on the device of a user, the GUI agent $\mathcal{F}$ receives a instruction x and captures the current device screenshot s_t. $\mathcal{F}$ then constructs a prompt that combines the screenshot s_t and the instruction x, which is subsequently used to predict an action a_t. Formally, at each time step t ($0 \leq t \leq T$), the process can be expressed as: $a_t = \mathcal{F}(s_t, x)$ where s_t is the screenshot at time step t, x is the instruction, and a_t is the action predicted by the agent at time step t.

Then this action a_t is executed, leading to a new screenshot s_{t+1}. $\mathcal{F}$ evaluates whether the instruction x has been completed. If x is not completed, the agent repeats this process until either x is completed or the maximum step limit T is reached. Throughout the execution process, it is possible that some pairs of screenshots and instructions (s_t, x) may deviate significantly from the distribution of $\mathcal{D}_{\text{ID}}$. Such pairs are classified as OOD samples. When these OOD samples are encountered, the agent's action predictions are prone to errors, which can lead to undesirable execution results.

Table 1: Results of the pilot study. Existing popular OOD detection methods perform poorly in the GUI agent domain.

	Smartphone		Computer		Web browser	
Method	AUROC ↑	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑	FPR95 ↓
Embedding-based methods						
TV score	54.26	98.02	60.13	85.37	60.27	89.48
Last layer embedding	50.94	76.48	64.31	68.73	67.51	74.58
Best layer embedding	76.14	48.92	89.72	45.60	89.77	31.75
Uncertainty-based methods						
Top-k confidence	67.07	85.10	57.51	93.59	57.40	94.41
Output entropy	67.07	92.69	57.51	86.22	57.40	86.73

The objective of OOD detection for GUI agents, therefore, is to identify whether the pair (s_t, x) at each time step t deviates from the distribution of $\mathcal{D}_{\text{ID}}$. If OOD is detected, the agent should immediately terminate the execution of the instruction and alert the user; otherwise, it continues with the execution. To formally define the OOD detection mechanism, we introduce the following OOD detection function f_{OOD}, which operates on each pair (s_t, x):

$$f_{\text{OOD}}(s_t, x) = \begin{cases} 1, & \text{if } (s_t, x) \text{ is OOD,} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

If $f_{\text{OOD}}(s_t, x)$ returns a value of 1, indicating that the current state are OOD, the agent will terminate the process and alert the user. Otherwise, the agent will predict the appropriate action a_t and proceed to execute it. The key to OOD detection for GUI agents is how to construct an effective $f_{\text{OOD}}(s_t, x)$.

3.2 Pilot Study with Popular OOD Detection Methods

The existing popular OOD detection methods [13, 14] can be broadly categorized into two main types: embedding-based methods [15, 16] and uncertainty-based methods [17, 18]. Based on existing research [54, 55] in OOD detection for (M)LLMs, we used three embedding-based methods and two uncertainty-based methods for experimentation. Detailed descriptions of these methods can be found in Appendix E.

We evaluated the five OOD detection methods used as baselines across eight datasets spanning smartphone, computer, and web browser platforms. Following standard OOD detection method evaluation criteria [56, 57], for each method, we plotted the ROC curve and reported the area under the receiver operating characteristic curve (AUROC) [58] and false positive rate at 95% true positive rate (FPR95) for each device category dataset.

The pilot experiment results are shown in Table 1. The AUROC metric reflects the separability between OOD and ID datasets achieved by different methods in the GUI agent domain. Even the best baseline performances across different platforms only range from 76.14% to 89.77%, barely reaching the acceptable level traditionally expected for OOD detection. The FPR95 metric measures the false positive rate when the true positive rate reaches 95%. Pilot experiments show that even the best baselines result in an FPR95 of 31.75% to 48.92%, indicating that while maintaining 95% task success for the GUI agent, 31.75% to 48.92% of OOD tasks would still be mistakenly executed, posing significant risks.

3.3 Why do these OOD detection methods perform suboptimally?

For embedding-based methods, due to the relatively uniform reasoning patterns of GUI agents—TV score (as shown in Figure 2), which rely on differences in layer embeddings to infer reasoning path divergence, perform suboptimally in OOD detection for GUI agents. Other methods based on single-layer embeddings have achieved better performance; however, since GUI agents tend to lose

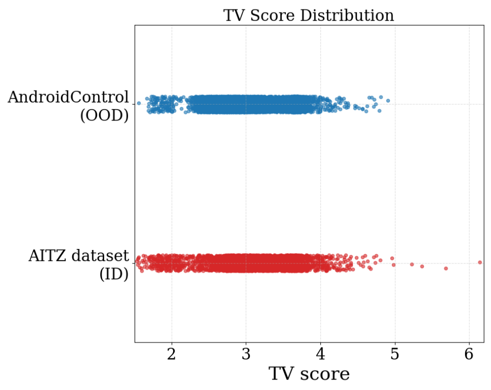

Figure 2: TV score distribution on AndroidControl dataset and AITZ dataset.

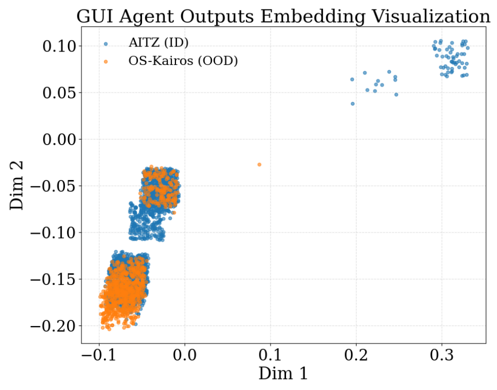

Figure 3: The output embeddings of the OS-Kairos dataset and AITZ dataset are visualized. Most of the sample points are confused together.

some information related to input understanding during the reasoning process, there is still room to improve.

As for uncertainty-based methods, the separability between the output spaces of ID and OOD samples in the GUI agent domain is extremely low (as shown in Figure 3). As a result, GUI agents struggle to estimate the uncertainty of their generated outputs, making uncertainty-based methods nearly ineffective at distinguishing between ID and OOD samples.

These OOD detection methods rely on finding a decision boundary by identifying the Youden Index [59] after obtaining some scoring metric, that is:

$$\text{Youden Index} = \arg\max_t \left(\text{TPR}(t) - \text{FPR}(t)\right), \tag{2}$$

where t is the threshold, and TPR and FPR represent the true positive rate and false positive rate at threshold t, respectively.

However, popular GUI agents such as OS-Atlas [33], UI-TARS [34], and AGUVIS [35] have access to abundant and diverse training data. In the embedding space, the ID dataset representations for these GUI agents naturally form clusters according to their different data sources. Moreover, we observe that even for datasets originating from a single data source such as AITZ [60], as shown in Figure 4, there can still be noticeable clustering phenomena in the embedding space. We also observe that samples farther from the centroid tend to yield higher action success rates (see Appendix B.2). For such non-linearly separable data, the Youden Index approach becomes inadequate. Instead, models like GMM, which are capable of fitting distributions in a clustered manner, are better suited for our scenario.

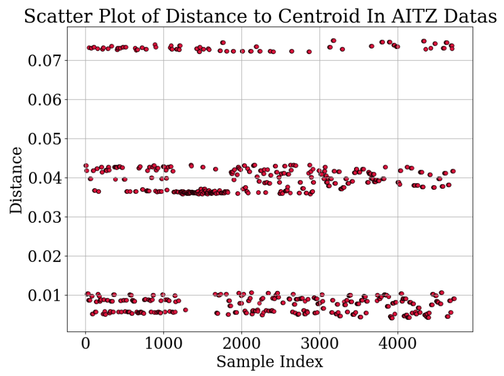

Figure 4: The multimodal embeddings of the AITZ dataset show a clustered distribution pattern around the centroid.

4 GEM Method

Pilot experiments in Section 3 show that popular OOD detection methods perform suboptimally in the GUI agent domain. However, we observe that the input embeddings generated by GUI agents naturally lend themselves to modeling the ID representation of $\mathcal{D}_{\text{ID}}$. The algorithm description can be found in Appendix F.

Given a GUI agent $\mathcal{F}$ and an ID dataset $\mathcal{D}_{\text{ID}} = \{(s_i, x_i)\}_{i=1}^k$, we first obtain an encoder layer l_e from $\mathcal{F}$. The encoder l_e maps each input pair (s_i, x_i) to an embedding vector $e_i \in \mathbb{R}^n$, formally:

$$e_i = l_e(s_i, x_i), \quad i = 1, \ldots, k. \tag{3}$$

Thus, we construct an ID embedding dataset $\mathcal{D}_{\text{embedding}} = \{e_i\}_{i=1}^k$. Each e_i is a point in the n-dimensional embedding space.

To model this distribution, we first compute the centroid μ of $\mathcal{D}_{\text{embedding}}$: $\mu = \frac{1}{k}\sum_{i=1}^k e_i$. Next, we calculate the Euclidean distance between each embedding e_i and the centroid μ:

$$d_i = \|e_i - \mu\|_2, \quad i = 1, \ldots, k, \tag{4}$$

resulting in a distance dataset $\mathcal{D}_{\text{distance}} = \{d_i\}_{i=1}^k$.

The distribution of $\mathcal{D}_{\text{distance}}$ is typically nonlinear and may contain multiple modes. Therefore, instead of fitting a simple Gaussian or applying heuristic thresholds, we model it using a GMM. Specifically, we assume that the distances are generated from a mixture of m univariate Gaussian components. The GMM models the probability density function as: $p(d) = \sum_{j=1}^m \pi_j \, \mathcal{N}(d \mid \mu_j, \sigma_j^2)$, π_j is the mixing coefficient of the j-th component, satisfying $\sum_{j=1}^m \pi_j = 1$ and $\pi_j \geq 0$, $\mathcal{N}(d \mid \mu_j, \sigma_j^2)$ denotes the density of a univariate Gaussian:

$$\mathcal{N}(d \mid \mu_j, \sigma_j^2) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{(d-\mu_j)^2}{2\sigma_j^2}\right). \tag{5}$$

Given the dataset $\mathcal{D}_{\text{distance}}$, the log-likelihood of the data under the GMM is:

$$\log \mathcal{L}_m = \sum_{i=1}^k \log\left(\sum_{j=1}^m \pi_j \, \mathcal{N}(d_i \mid \mu_j, \sigma_j^2)\right). \tag{6}$$

The parameters $\{\pi_j, \mu_j, \sigma_j^2\}$ are estimated by maximizing the log-likelihood $\log \mathcal{L}_m$ via the expectation-maximization (EM) algorithm. Each iteration of EM consists of E-step and M-step. E-step estimate the posterior probability that sample d_i belongs to component j:

$$\gamma_{ij} = \frac{\pi_j \, \mathcal{N}(d_i \mid \mu_j, \sigma_j^2)}{\sum_{l=1}^m \pi_l \, \mathcal{N}(d_i \mid \mu_l, \sigma_l^2)}. \tag{7}$$

M-step update the parameters based on the estimated responsibilities:

$$\pi_j^{\text{new}} = \frac{1}{k}\sum_{i=1}^k \gamma_{ij}, \quad \mu_j^{\text{new}} = \frac{\sum_{i=1}^k \gamma_{ij} d_i}{\sum_{i=1}^k \gamma_{ij}}, \quad \sigma_j^{2\,\text{new}} = \frac{\sum_{i=1}^k \gamma_{ij}(d_i - \mu_j^{\text{new}})^2}{\sum_{i=1}^k \gamma_{ij}}. \tag{8}$$

The E-step and M-step are alternated until convergence to a local maximum of the likelihood.

To determine the optimal number of components m, we employ the BIC, defined as:

$$\text{BIC}(m) = -2\log \mathcal{L}_m + m \log k, \tag{9}$$

The optimal number of components m^* is thus selected as $m^* = \arg\min_m \text{BIC}(m)$.

Once the GMM is fitted with m^* components, each Gaussian component $\mathcal{N}(\mu_j, \sigma_j^2)$ provides a center μ_j and a standard deviation σ_j. We define the ID boundary for each component as the interval $[\mu_j - n\sigma_j, \mu_j + n\sigma_j]$. The details regarding the choice of n can be found in Appendix G.

At inference time, given a new input pair (s_t, x), the GUI agent computes the embedding $e_t = l_e(s_t, x)$ and its distance to the centroid $d_t = \|e_t - \mu\|_2$.

To determine whether (s_t, x) is ID, we check whether d_t falls within any of the ID boundaries defined by the fitted GMM components. Formally, the OOD detection function $f_{\text{OOD}}(s_t, x)$ is given by:

$$f_{\text{OOD}}(s_t, x) = \begin{cases} 0, & \text{if } \exists j \text{ such that } d_t \in [\mu_j - n\sigma_j, \mu_j + n\sigma_j], \\ 1, & \text{otherwise.} \end{cases} \tag{10}$$

If $f_{\text{OOD}}(s_t, x) = 1$, the input is classified as OOD, and the agent immediately terminates execution and notifies the user. Otherwise, the agent proceeds with its normal action prediction and execution.

5 Experiments

In this section, we first introduce the experimental setup, followed by a presentation and analysis of the performance of GEM on OOD detection for GUI agents.

5.1 Experiments Setup

Datasets. For the ID dataset, we use the AITZ [60] dataset, which contains GUI agent data covering more than 70 Android app scenarios. For the OOD datasets, we select 8 GUI agent datasets spanning three platforms: smartphone, computer, and web browser. The smartphone platform includes AndroidControl [61], OS-Kairos [62], Meta-GUI [63], and ScreenSpot-Mobile [64],while the computer platform includes Omniact-Desktop [65] and ScreenSpot-Desktop, and the web browser platform includes Omniact-Web and ScreenSpot-Web. The four smartphone datasets simulate extrapolation-OOD scenarios, whereas the computer and web browser datasets simulate internalization-OOD scenarios. Detailed description of the datasets is provided in Appendix C.

Implementation. Popular GUI Agents [33, 34, 35] are developed based on Qwen2-VL-7B. To simulate the construction process of a popular GUI Agent, we perform SFT on Qwen2-VL-7B using AITZ train dataset, and then evaluate its OOD detection performance on each sample from eight different OOD datasets and AITZ test dataset (ID). We report the accuracy, precision, recall, and F1 score for each dataset. Detailed descriptions of the experiments is provided in Appendix D.

Baseline. We use two commonly-used types of OOD detection methods as our experimental baselines: embedding-based methods and uncertainty-based methods. The embedding-based methods include the TV score [21], last layer embedding, and best layer embedding. The uncertainty-based methods include top-k confidence and output entropy. More detailed descriptions and formal definitions of these baseline methods are provided in Appendix E.

5.2 Main Results

Table 2 shows the main results. GEM consistently outperforms existing methods across almost all datasets. Although on the Omniact-Desktop dataset the accuracy is 1.62% lower than the best-performing baseline, our method achieves substantial improvements in other metrics. This shows a better balance between rejecting OOD samples and retaining ID samples, which is critical for reliable OOD detection. On other seven datasets, GEM outperforms all baselines across all evaluation metrics.

Because GUI agents exhibit relatively simple reasoning paths, limiting the effectiveness of methods like the TV score, which rely on modeling diverse reasoning paths. And the complex semantics of multimodal inputs weakens the performance of other embedding-based approaches. Furthermore, due to the semantic similarity in the output space, uncertainty-based OOD detection methods for GUI agents are unreliable. In contrast, GEM effectively captures subtle high-dimensional differences between ID and OOD data, leading to strong and robust detection.

6 Further analysis

In this section, we present several interesting observations based on our proposed method as well as traditional OOD detection methods.

6.1 Generalization Experiment

To demonstrate the generalization ability of GEM, we evaluate its performance across nine different GUI agents or MLLMs. As shown in Table 3, GEM consistently achieves high OOD detection accuracy across five models spanning three platforms, indicating its strong generalization capability.

For the other four models where GEM performs suboptimally, the results remain explainable. Specifically, OS-Atlas-Base-7B and OS-Atlas-Pro-7B were exposed to most of the smartphone datasets used in our experiment during their GUI grounding pretraining phase, and their visual encoders were not frozen, which contributed to their weaker OOD performance on the smartphone platform. Meanwhile, LLaVA-1.5 and BLIP capture visual information at a coarser granularity. Although they still achieve

Table 2: Comparison of OOD Detection Results Across Different Datasets. For each metric, the best result among baseline methods is underlined.

Metric	AndroidControl				OS-Kairos			
Method	Acc.	Prec.	Rec.	F1	Acc.	Prec.	Rec.	F1
TV score	55.37	42.48	68.95	52.57	77.36	90.20	82.20	86.01
Top-k confidence	68.29	77.30	71.58	74.33	63.71	26.04	74.47	38.59
Output entropy	68.29	55.13	62.43	58.55	63.71	93.05	61.77	74.25
Last layer embedding	70.08	69.02	96.77	80.57	32.66	17.45	91.10	29.29
Best layer embedding	78.20	77.02	94.07	84.69	74.35	33.29	67.33	44.56
GEM (ours)	**99.39**	**98.33**	**100.0**	**99.16**	**100.0**	**100.0**	**100.0**	**100.0**
△ (**GEM** - underline)	+21.19	+21.03	+3.23	+14.47	+22.64	+6.95	+8.90	+13.99

Metric	Meta-GUI				ScreenSpot-Mobile			
Method	Acc.	Prec.	Rec.	F1	Acc.	Prec.	Rec.	F1
TV score	68.56	71.91	91.51	80.53	65.54	92.01	67.76	78.04
Top-k confidence	54.60	34.54	63.60	44.77	52.93	11.91	60.96	19.92
Output entropy	54.60	77.46	50.93	61.46	52.93	92.62	52.07	66.67
Last layer embedding	59.38	39.47	75.72	51.89	30.71	12.05	98.61	21.47
Best layer embedding	76.59	55.50	96.31	70.42	62.72	19.60	92.83	32.36
GEM (ours)	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
△ (**GEM** - underline)	+23.41	+22.54	+3.69	+19.47	+34.46	+7.38	+1.39	+21.96

Metric	Omniact-Desktop				ScreenSpot-Desktop			
Method	Acc.	Prec.	Rec.	F1	Acc.	Prec.	Rec.	F1
TV score	47.50	29.04	82.36	42.94	23.92	7.66	95.21	14.18
Top-k confidence	44.54	84.05	33.36	47.76	31.00	95.43	27.43	42.62
Output entropy	44.54	27.47	79.95	40.88	31.00	7.35	81.44	13.49
Last layer embedding	60.05	35.80	83.84	50.17	46.54	10.05	89.22	18.06
Best layer embedding	91.15	83.73	78.34	80.94	43.46	10.18	96.71	18.43
GEM (ours)	**89.53**	**87.89**	**100.0**	**93.55**	**96.86**	**96.74**	**100.0**	**98.34**
△ (**GEM** - underline)	-1.62	+3.84	+16.16	+12.61	+50.32	+1.14	+3.29	+55.63

Metric	Omniact-Web				ScreenSpot-Web			
Method	Acc.	Prec.	Rec.	F1	Acc.	Prec.	Rec.	F1
TV score	48.63	13.07	72.45	22.15	45.35	11.27	79.59	19.75
Top-k confidence	61.36	91.17	63.15	74.61	42.97	94.91	39.84	56.12
Output entropy	61.36	12.16	45.47	19.19	42.97	10.54	76.83	18.54
Last layer embedding	61.82	18.52	81.89	30.20	47.77	12.64	87.61	22.09
Best layer embedding	86.03	40.34	80.38	53.72	74.77	24.77	97.48	39.50
GEM (ours)	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
△ (**GEM** - underline)	+13.97	+8.83	+18.11	+25.39	+25.23	+5.09	+2.52	+43.88

respectable OOD detection accuracy (72.44% to 88.38%) in internalization-OOD scenarios involving the computer and web browser platforms, their performance degrades when both the ID and OOD datasets are from the smartphone domain. This highlights their limitations in handling the complex embedding space specific to GUI agent tasks. This also indicates that extrapolation-OOD scenarios are more challenging for OOD detection than internalization-OOD scenarios.

We also demonstrate the rationality and stability of GEM through ablation studies (see Appendix G).

6.2 Layer-Level OOD Detection Analysis of GUI Agents

We evaluated the effectiveness of representations extracted from each of the twenty-eight layers of Qwen2-VL-7B for OOD detection using an embedding-based approach.

Table 3: The performance of GEM with different encoder structures.

Model	Smartphone		Computer		Web browser	
	Accuary	F1 score	Accuary	F1 score	Accuary	F1 score
UI-TARS-7B	97.94	96.55	83.22	89.58	98.28	98.97
Qwen2-VL-2B	98.64	97.68	88.41	92.56	100.0	100.0
Qwen2-VL-7B	99.51	99.16	87.63	92.10	100.0	100.0
Qwen2.5-VL-3B	93.77	90.22	88.44	92.58	100.0	100.0
Qwen2.5-VL-7B	99.78	99.61	89.97	93.50	100.0	100.0
OS-Atlas-Base-7B	33.08	46.19	74.29	84.87	83.02	90.72
OS-Atlas-Pro-7B	33.08	46.19	74.29	84.87	83.02	90.72
LLaVA1.5	28.83	44.66	72.44	83.96	83.11	90.77
Blip + BERT	30.04	45.06	77.81	86.66	88.38	93.45

As shown in Figure 5, in most datasets, the AUROC first increases as the layer depth increases, then decreases. However, the AUROC rises again in the last two layers. Interestingly, in 4 out of 8 datasets, the best OOD detection results are achieved at the ninth layer. This might be because, as the layers get deeper, the importance of specific task-related features increases, while the contribution of general visual or textual features decreases. Around the ninth layer, the GUI agent likely finds a balance between these two factors. The increase in AUROC in the final two layers might be because, at this stage, the GUI agent has developed a kind of confidence estimation in its final output, allowing it to better distinguish between OOD and ID samples.

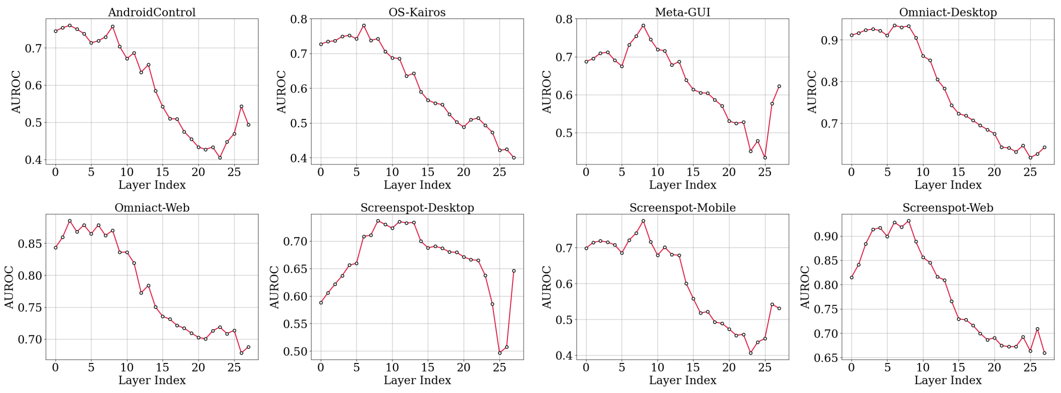

Figure 5: AUROC for OOD detection using embeddings from different layers of the GUI agent.

7 Conclusion

We present the first systematic analysis of OOD detection for GUI agents and show that traditional OOD detection methods perform suboptimally in this domain. We find that the multimodal inputs of GUI agents exhibit a clustered pattern in the semantic space based on their distance from the centroid. Based on this, we propose GEM, a method that uses input embedding information to fit a GMM to detect OOD samples for GUI agents. Experiments on eight datasets across three platforms demonstrate the superior performance of GEM over all baselines. Generalization experiments on nine different GUI agents or MLLM backbones validate the excellent generalization ability of GEM. We also find that for the GUI agent built with Qwen2-VL-7B, semantic information around the ninth layer achieves the best OOD detection performance.

References

[1] 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.

[2] Shuai Wang, Weiwen Liu, Jingxuan Chen, Weinan Gan, Xingshan Zeng, Shuai Yu, Xinlong Hao, Kun Shao, Yasheng Wang, and Ruiming Tang. Gui agents with foundation models: A comprehensive survey. *arXiv preprint arXiv:2411.04890*, 2024.

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

[4] William Liu, Liang Liu, Yaxuan Guo, Han Xiao, Weifeng Lin, Yuxiang Chai, Shuai Ren, Xiaoyu Liang, Linghao Li, Wenhao Wang, et al. Llm-powered gui agents in phone automation: Surveying progress and prospects. *arXiv preprint arXiv:2412.13501*, 2025.

[5] Yucheng Shi, Wenhao Yu, Wenlin Yao, Wenhu Chen, and Ninghao Liu. Towards trustworthy gui agents: A survey. *arXiv preprint arXiv:2503.23434*, 2025.

[6] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pages 19730–19742. PMLR, 2023.

[7] Wenhui Wang, Hangbo Bao, Li Dong, Johan Bjorck, Zhiliang Peng, Qiang Liu, Kriti Aggarwal, Owais Khan Mohammed, Saksham Singhal, Subhojit Som, et al. Image as a foreign language: Beit pretraining for vision and vision-language tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19175–19186, 2023.

[8] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[9] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[10] Zhuosheng Zhang, Aston Zhang, Mu Li, George Karypis, Alex Smola, et al. Multimodal chain-of-thought reasoning in language models. *Transactions on Machine Learning Research*, 2023.

[11] Yuhang Liu, Pengxiang Li, Zishu Wei, Congkai Xie, Xueyu Hu, Xinchen Xu, Shengyu Zhang, Xiaotian Han, Hongxia Yang, and Fei Wu. Infiguiagent: A multimodal generalist gui agent with native reasoning and reflection. *arXiv preprint arXiv:2501.04575*, 2025.

[12] Zhiyuan Hu, Shiyun Xiong, Yifan Zhang, See-Kiong Ng, Anh Tuan Luu, Bo An, Shuicheng Yan, and Bryan Hooi. Guiding vlm agents with process rewards at inference time for gui navigation. *arXiv preprint arXiv:2504.16073*, 2025.

[13] Andrey Malinin and Mark Gales. Predictive uncertainty estimation via prior networks. *Advances in neural information processing systems*, 31, 2018.

[14] Jingkang Yang, Kaiyang Zhou, Yixuan Li, and Ziwei Liu. Generalized out-of-distribution detection: A survey. *International Journal of Computer Vision*, 132(12):5635–5662, 2024.

[15] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *Advances in neural information processing systems*, 31, 2018.

[16] Yiyou Sun, Yifei Ming, Xiaojin Zhu, and Yixuan Li. Out-of-distribution detection with deep nearest neighbors. In *International Conference on Machine Learning*, pages 20827–20840. PMLR, 2022.

[17] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

[18] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[19] Jie Ren, Jiaming Luo, Yao Zhao, Kundan Krishna, Mohammad Saleh, Balaji Lakshminarayanan, and Peter J Liu. Out-of-distribution detection and selective generation for conditional language models. *arXiv preprint arXiv:2209.15558*, 2022.

[20] Corentin Kervadec, Grigory Antipov, Moez Baccouche, and Christian Wolf. Roses are red, violets are blue... but should vqa expect them to? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2776–2785, 2021.

[21] Yiming Wang, Pei Zhang, Baosong Yang, Derek Wong, Zhuosheng Zhang, and Rui Wang. Embedding trajectory for out-of-distribution detection in mathematical reasoning. *Advances in Neural Information Processing Systems*, 37:42965–42999, 2024.

[22] Qianhui Wu, Huiqiang Jiang, Haonan Yin, Börje F Karlsson, and Chin-Yew Lin. Multi-level knowledge distillation for out-of-distribution detection in text. *arXiv preprint arXiv:2211.11300*, 2022.

[23] Xinbei Ma, Zhuosheng Zhang, and Hai Zhao. CoCo-agent: A comprehensive cognitive MLLM agent for smartphone GUI automation. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 9097–9110, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[24] Wenjia Jiang, Yangyang Zhuang, Chenxi Song, Xu Yang, Joey Tianyi Zhou, and Chi Zhang. Appagentx: Evolving gui agents as proficient smartphone users. *arXiv preprint arXiv:2503.02268*, 2025.

[25] Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. Mobile-agent-e: Self-evolving mobile assistant for complex tasks. *arXiv preprint arXiv:2501.11733*, 2025.

[26] Guangyi Liu, Pengxiang Zhao, Liang Liu, Zhiming Chen, Yuxiang Chai, Shuai Ren, Hao Wang, Shibo He, and Wenchao Meng. Learnact: Few-shot mobile gui agent with a unified demonstration benchmark. *arXiv preprint arXiv:2504.13805*, 2025.

[27] Douglas A Reynolds et al. Gaussian mixture models. *Encyclopedia of biometrics*, 741(659-663):3, 2009.

[28] Andrew A Neath and Joseph E Cavanaugh. The bayesian information criterion: background, derivation, and applications. *Wiley Interdisciplinary Reviews: Computational Statistics*, 4(2):199–203, 2012.

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

[30] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

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

[32] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14281–14290, 2024.

[33] 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.

[34] Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

[35] Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*, 2024.

[36] Yanda Li, Chi Zhang, Wanqi Yang, Bin Fu, Pei Cheng, Xin Chen, Ling Chen, and Yunchao Wei. Appagent v2: Advanced agent for flexible mobile interactions. *arXiv preprint arXiv:2408.11824*, 2024.

[37] Zongru Wu, Pengzhou Cheng, Zheng Wu, Tianjie Ju, Zhuosheng Zhang, and Gongshen Liu. Smoothing grounding and reasoning for mllm-powered gui agents with query-oriented pivot tasks. *arXiv preprint arXiv:2503.00401*, 2025.

[38] Zhuosheng Zhang and Aston Zhang. You only look at screens: Multimodal chain-of-action agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3132–3149, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[39] Qinzhuo Wu, Wei Liu, Jian Luan, and Bin Wang. ReachAgent: Enhancing mobile agent via page reaching and operation. In Luis Chiruzzo, Alan Ritter, and Lu Wang, editors, *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4760–4775, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics.

[40] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, et al. Tool learning with foundation models. *ACM Computing Surveys*, 57(4):1–40, 2024.

[41] Yifei Zhou, Hao Bai, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. In *Automated Reinforcement Learning: Exploring Meta-Learning, AutoML, and LLMs*, 2024.

[42] Xiao Liu, Bo Qin, Dongzhu Liang, Guang Dong, Hanyu Lai, Hanchen Zhang, Hanlin Zhao, Iat Long Iong, Jiadai Sun, Jiaqi Wang, et al. Autoglm: Autonomous foundation agents for guis. *arXiv preprint arXiv:2411.00820*, 2024.

[43] Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, et al. Autowebglm: A large language model-based web navigating agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5295–5306, 2024.

[44] Taiyi Wang, Zhihao Wu, Jianheng Liu, Derek Yuen, HAO Jianye, Jun Wang, and Kun Shao. Distrl: An asynchronous distributed reinforcement learning framework for on-device control agent. In *NeurIPS 2024 Workshop on Fine-Tuning in Modern Machine Learning: Principles and Scalability*, 2024.

[45] Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Guanjing Xiong, and Hongsheng Li. Ui-r1: Enhancing action prediction of gui agents by reinforcement learning. *arXiv preprint arXiv:2503.21620*, 2025.

[46] Xiaobo Xia and Run Luo. Gui-r1: A generalist r1-style vision-language action model for gui agents. *arXiv preprint arXiv:2504.10458*, 2025.

[47] Pengzhou Cheng, Zheng Wu, Zongru Wu, Aston Zhang, Zhuosheng Zhang, and Gongshen Liu. Os-kairos: Adaptive interaction for mllm-powered gui agents. *arXiv preprint arXiv:2503.16465*, 2025.

[48] Chaoran Chen, Zhiping Zhang, Bingcan Guo, Shang Ma, Ibrahim Khalilov, Simret A Gebreegziabher, Yanfang Ye, Ziang Xiao, Yaxing Yao, Tianshi Li, et al. The obvious invisible threat: Llm-powered gui agents' vulnerability to fine-print injections. *arXiv preprint arXiv:2504.11281*, 2025.

[49] Jungjae Lee, Dongjae Lee, Chihun Choi, Youngmin Im, Jaeyoung Wi, Kihong Heo, Sangeun Oh, Sunjae Lee, and Insik Shin. Safeguarding mobile gui agent via logic-based action verification. *arXiv preprint arXiv:2503.18492*, 2025.

[50] Weitang Liu, Xiaoyun Wang, John Owens, and Yixuan Li. Energy-based out-of-distribution detection. *Advances in neural information processing systems*, 33:21464–21475, 2020.

[51] Jie Ren, Peter J Liu, Emily Fertig, Jasper Snoek, Ryan Poplin, Mark Depristo, Joshua Dillon, and Balaji Lakshminarayanan. Likelihood ratios for out-of-distribution detection. *Advances in neural information processing systems*, 32, 2019.

[52] Yifei Ming, Ziyang Cai, Jiuxiang Gu, Yiyou Sun, Wei Li, and Yixuan Li. Delving into out-of-distribution detection with vision-language representations. *Advances in neural information processing systems*, 35:35087–35102, 2022.

[53] Hualiang Wang, Yi Li, Huifeng Yao, and Xiaomeng Li. Clipn for zero-shot ood detection: Teaching clip to say no. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1802–1812, 2023.

[54] Hao Dong, Yue Zhao, Eleni Chatzi, and Olga Fink. Multiood: Scaling out-of-distribution detection for multiple modalities. *arXiv preprint arXiv:2405.17419*, 2024.

[55] Shuo Lu, Yingsheng Wang, Lijun Sheng, Aihua Zheng, Lingxiao He, and Jian Liang. Recent advances in ood detection: Problems and approaches. *arXiv preprint arXiv:2409.11884*, 2024.

[56] Siyu Luan, Zonghua Gu, Leonid B Freidovich, Lili Jiang, and Qingling Zhao. Out-of-distribution detection for deep neural networks with isolation forest and local outlier factor. *IEEE Access*, 9:132980–132989, 2021.

[57] Peng Cui and Jinjia Wang. Out-of-distribution (ood) detection based on deep learning: A review. *Electronics*, 11(21):3500, 2022.

[58] Charles E Metz. Basic principles of roc analysis. *Seminars in Nuclear Medicine*, 8(4):283–298, 1978.

[59] Ronen Fluss, David Faraggi, and Benjamin Reiser. Estimation of the youden index and its associated cutoff point. *Biometrical Journal: Journal of Mathematical Methods in Biosciences*, 47(4):458–472, 2005.

[60] Jiwen Zhang, Jihao Wu, Teng Yihua, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu Tang. Android in the zoo: Chain-of-action-thought for GUI agents. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 12016–12031, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[61] Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William E Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Kenji Toyama, Robert James Berry, Divya Tyamagundlu, Timothy P Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents. In *The Thirteenth International Conference on Learning Representations*, 2025.

[62] Pengzhou Cheng, Zheng Wu, Zongru Wu, Aston Zhang, Zhuosheng Zhang, and Gongshen Liu. Os-kairos: Adaptive interaction for mllm-powered gui agents. *arXiv preprint arXiv:2503.16465*, 2025.

[63] Liangtai Sun, Xingyu Chen, Lu Chen, Tianle Dai, Zichen Zhu, and Kai Yu. Meta-gui: Towards multi-modal conversational agents on mobile gui. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6699–6712, 2022.

[64] Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. Screenspot-pro: Gui grounding for professional high-resolution computer use. *arXiv preprint arXiv:2504.07981*, 2025.

[65] 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 *European Conference on Computer Vision*, pages 161–178. Springer, 2024.

A Ethics Statement

All datasets and models used in this work are sourced from the official repositories associated with the original papers, and we strictly follow their respective usage protocols. The datasets remain unmodified, and the models are only subjected to supervised fine-tuning and inference on OOD data. To ensure the safety of model outputs, all generated results have been manually reviewed to prevent any potentially harmful or inappropriate content. As our research focuses exclusively on OOD detection for GUI agents and does not involve sensitive or personal data, we believe our work poses minimal risk of societal harm.

B Expanded Analysis

In this section, we provide an expanded analysis of content not fully covered in the main text. First, we discuss our limitations, and then we present a detailed analysis of the clustering phenomenon in the input semantic space of the GUI agent.

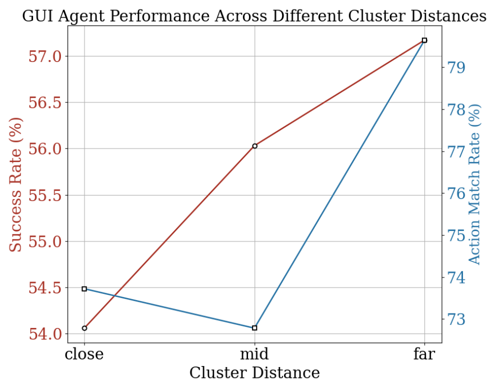

Figure 6: GUI agent performance across different cluster distances.

B.1 Limitations

Due to the evolving environments of GUI agent domain, although our experiments cover major datasets across three platforms—smartphones, computers, and web browsers—there is still a gap between our setup and the simulation of real-world GUI agent environments.

GEM adopts a GMM to fit the distribution of multimodal input semantics in the ID dataset. Although GEM achieves state-of-the-art performance on mainstream benchmark datasets, its underlying assumption—that multimodal semantic distributions can be effectively modeled using a limited number of Gaussian components—may not hold under highly anomalous GUI data distributions, where the semantic space exhibits irregular or complex patterns that are difficult to approximate with a finite mixture model.

B.2 The detailed analysis of clustering phenomenon

We further evaluate the agent's performance across different clusters on the AITZ dataset. s shown in Figure 6, we also observe that the farther a sample is scattered from the centroid, the higher the action success rate of the GUI agent tends to be. Interestingly, although the action match rate slightly drops at mid-range distances, it increases significantly again at far distances. This is because, in the embedding hypersphere, samples that lie farther from the centroid are less likely to be confused with others, making their knowledge more distinctly learnable by the model.

C Datasets Details

In this section, we provide a detailed description to the datasets used in this paper.

• **AITZ** is the first dataset to annotate Chain-of-Action-Thought (CoAT) for GUI agents, which considers descriptions of prior actions and the current screen, the actions that should be taken, and reasoning about the potential consequences of the selected actions. It contains 18,643 screen-action pairs with CoAT annotations.

• **AndroidControl** is a dataset covering 833 distinct Android applications and containing 15,283 everyday tasks performed within these apps. This dataset is currently the most diverse benchmark for computer-based control, providing a foundation for in-depth model performance analysis.

• **OS-Kairos** is a dataset containing data from 12 different Chinese and English mobile applications across 12 topic categories. The tasks in this dataset were automatically collected by an agent capability probing framework and later curated by humans. It covers a wide range of scenarios encountered in everyday life.

• **Meta-GUI** is a dataset containing dialogues and GUI interaction trajectories. It covers six high-frequency daily-life topics across 11 mobile applications.

• **Omniact** is a dataset consisting of 9.8K pairs of screen images and natural language task descriptions, covering a variety of operating systems and web applications. The dataset includes tasks of varying difficulty levels, ranging from simple single-step actions to complex multi-step operations.

• **ScreenSpot** is a dataset containing high-resolution real screenshots and expert annotations from various domains. This dataset covers data from multiple platforms, including Mac, Windows, Android, iPhone, and web, making it the most platform-diverse GUI dataset.

D Main Experiment Details

In this section, we provide more details about our main experiment.

Hyperparameter Settings. All random seeds in our main experiment are set to 42. The number of GMM clusters selected by BIC ranges from 1 to 15. During SFT, we train for 7 epochs with a batch size of 2 and a learning rate of 1.0e-5. The SFT process takes approximately 3.6 hours on 4 A100 GPUs (80GB each).

Evaluation Metrics Details. After performing SFT on Qwen2-VL-7B using the AITZ training set, we fit a distance-based GMM on the same training set following the GEM method. Classification boundaries are defined using three standard deviations per cluster. The accuracy, precision, recall, and F1 scores reported in Table 2 are obtained by treating each dataset as the OOD samples and the AITZ test set as the ID samples. The prompt of GEM and all baselines is shown in Figure 7

E Baseline Details

In this section, we provide a detailed description to the baseline methods used in this paper.

E.1 TV score

TV score is an OOD detection method that has been proven effective in the domain of mathematical reasoning with LLMs. The TV score measures the instability or variation of a test sample's layer-wise hidden representations with respect to a Gaussian distribution fitted to ID samples. The computation proceeds as follows: Let $\hat{y}_l^{(i)}$ represent the hidden representations of the i-th ID sample at layer l, where l ranges from 1 to L and i ranges from 1 to N. Let y_l represent the hidden representations of the test sample being evaluated at layer l, where l ranges from 1 to L.

For each layer l, we estimate a Gaussian distribution $\mathcal{G}_l = \mathcal{N}(\mu_l, \Sigma_l)$ by computing the empirical mean μ_l and covariance Σ_l of the ID embeddings at that layer. The out-of-distribution (OOD) score for the evaluated sample at layer l is then defined as the Mahalanobis distance:

$$f(\mathbf{y}_l) = (\mathbf{y}_l - \mu_l)^\top \Sigma_l^{-1} (\mathbf{y}_l - \mu_l). \tag{11}$$

To assess higher-order fluctuations in the layer-wise representation trajectory, we further define a smoothed i-th order differential form of the representation. For each differential order $i = 1, \ldots, k$, we compute transformed Gaussian parameters:

$$\mu_l^{(i)} = \sum_{t=0}^{i} (-1)^{i+t} \binom{i}{t} \mu_{l+t}, \quad \Sigma_l^{(i)} = \sum_{t=0}^{i} \binom{i}{t} \Sigma_{l+t}. \tag{12}$$

Using these, we compute the Mahalanobis distance for the i-th order differential:

$$f^{(i)}(\mathbf{y}_l) = (\mathbf{y}_l^{(i)} - \mu_l^{(i)})^\top (\Sigma_l^{(i)})^{-1} (\mathbf{y}_l^{(i)} - \mu_l^{(i)}), \tag{13}$$

Prompt Template

```
You are now operating in Executable Language Grounding mode. Your goal is to help users accomplish tasks by suggesting executable actions that best fit their needs.
Your skill set includes both basic and custom actions:

1. Basic Actions
    Basic actions are standardized and available across all platforms. They provide essential functionality and are defined with a specific format, ensuring consistency and reliability.
    Basic Action 1: CLICK
        - purpose: Click at the specified position.
        - format: CLICK <point>[[x-axis, y-axis]]</point>
        - example usage: CLICK <point>[[101, 872]]</point>

    Basic Action 2: TYPE
        - purpose: Enter specified text at the designated location.
        - format: TYPE [input text]
        - example usage: TYPE [Shanghai shopping mall]

    Basic Action 3: SCROLL
        - Purpose: SCROLL in the specified direction.
        - Format: SCROLL [direction (UP/DOWN/LEFT/RIGHT)]
        - Example Usage: SCROLL [UP]


2. Custom Actions
    Custom actions are unique to each user's platform and environment. They allow for flexibility and adaptability, enabling the model to support new and unseen actions defined by users. These actions extend the functionality of the basic set, making the model more versatile and capable of handling specific tasks.
    Custom Action 1: PRESS_BACK
        - purpose: Press a back button to navigate to the previous screen.
        - format: PRESS_BACK
        - example usage: PRESS_BACK

    Custom Action 2: PRESS_HOME
        - purpose: Press a home button to navigate to the home page.
        - format: PRESS_HOME
        - example usage: PRESS_HOME

    Custom Action 3: COMPLETE
        - purpose: Indicate the task is finished.
        - format: COMPLETE
        - example usage: COMPLETE

    Custom Action 4: IMPOSSIBLE
        - purpose: Indicate the task is impossible.
        - format: IMPOSSIBLE
        - example usage: IMPOSSIBLE

    And your current task instruction and associated screenshot are as follows:
    Final goal: {obs['task']}
    Screenshot:
    Your output must be in one line. Do not split it into two lines.
    Your output must strictly follow the format below, and especially avoid using unnecessary quotation marks or other punctuation marks:
    action:
```

Figure 7: Prompt for GEM and all baselines.

where $\mathbf{y}_l^{(i)}$ denotes the i-th order smoothed difference of the sample's layer-wise representations, constructed analogously to the means.

Finally, the TV score at differential order i is obtained by averaging the per-layer scores:

$$\mathrm{TV}_i = \frac{1}{L} \sum_{l=1}^{L} f^{(i)}(\mathbf{y}_l). \tag{14}$$

E.2 Top-k confidence

Top-k confidence is a likelihood-based OOD detection method that evaluates the probability assigned by the language model to its most confident completions. Given a test input, the model generates k candidate output sequences, each consisting of a sequence of tokens. Let $P_j = \prod_{t=1}^{T_j} p(y_t^{(j)} \mid y_{<t}^{(j)}, x)$ denote the joint probability of the j-th output sequence, where x is the input, $y_t^{(j)}$ is the t-th token in the j-th output, and T_j is the output length. The Top-k confidence score is then defined as:

$$\text{Top-k} = \max_{j \in 1,\ldots,k} P_j. \tag{15}$$

A lower Top-k confidence score indicates a higher likelihood that the input is out-of-distribution, as the model fails to assign high probability to any of its top candidates.

E.3 Output entropy

Output entropy captures the model's uncertainty over the space of generated sequences and provides a distributional measure of output dispersion. For the same k candidate sequences used in Top-k confidence, let P_j denote the joint probability of the j-th output sequence. The normalized distribution over candidates is given by:

$$\tilde{P}_j = \frac{P_j}{\sum_{i=1}^{k} P_i}. \tag{16}$$

The entropy of this distribution is then computed as:

$$\text{Entropy} = -\sum_{j=1}^{k} \tilde{P}_j \log \tilde{P}_j. \tag{17}$$

Higher entropy suggests that the model is more uncertain about its output space, and such uncertainty often correlates with inputs being OOD.

E.4 Last layer embedding

This method assesses distributional proximity in the feature space of the model's final layer. Let $\mathbf{y}_L$ denote the representation of the test sample at the final layer L, and let $\{\hat{y}_L^{(i)}\}_{i=1}^{N}$ be the final-layer representations of the N in-distribution (ID) samples. We compute the empirical mean of the ID embeddings as:

$$\mu_L = \frac{1}{N} \sum_{i=1}^{N} \hat{y}_L^{(i)}. \tag{18}$$

The OOD score is then defined as the Euclidean distance from the test sample's final-layer representation to this mean:

$$f(y_L) = |y_L - \mu_L|_2. \tag{19}$$

A larger distance indicates that the sample lies further from the ID cluster in representation space, suggesting a higher likelihood of being OOD.

E.5 Best layer embedding

Best layer embedding extends the previous approach by selecting the most discriminative layer for OOD detection. For each layer $l \in \{1, \ldots, L\}$, we compute the per-layer representation $\mathbf{y}_l$ of the test

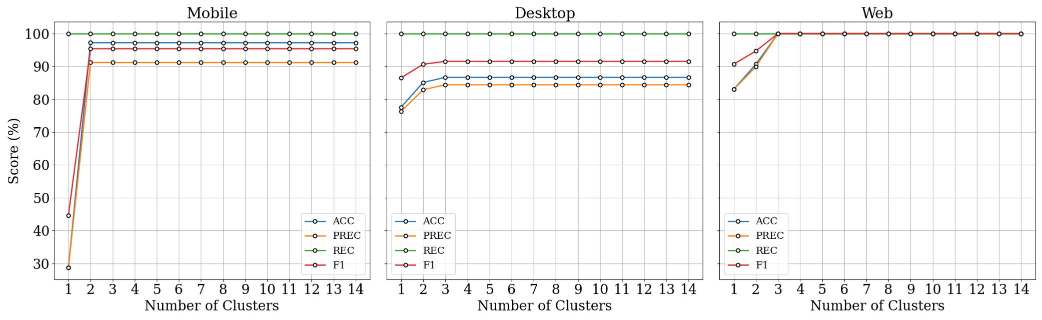

Figure 8: Ablation study on the maximum number of clusters.

sample and the corresponding ID mean μ_l using the ID samples. The distance is computed similarly as:

$$f_l(y_l) = |y_l - \mu_l|_2. \tag{20}$$

To determine the best layer l^*, we evaluate the OOD detection performance of each layer using a held-out validation set and select the one achieving the highest AUROC. The final score is then:

$$\text{BestLayerScore} = f_{l^*}(y_{l^*}). \tag{21}$$

This method allows flexibility in choosing the most informative representation space for distinguishing OOD inputs.

F GEM Algorithm

In this section, we present the algorithm of our method in Algorithm 1.

G Ablation Study

In this section, we conduct a ablation study on several adjustable parameters of the GEM algorithm. In all experiments presented in the main text, the number of clusters m for BIC search ranges from 1 to 10, and each GMM cluster defines classification boundaries based on three standard deviations. In this section, we perform an ablation study focusing on the number of clusters m used in BIC search and the granularity of the classification boundary. As shown in Figure 8, we set the maximum number of clusters for the BIC search from 1 to 15. We then report the performance of GEM on mobile, desktop, and web platforms under different cluster number ranges. We observe that once the number of clusters reaches a certain threshold, further increases have negligible impact on performance, indicating that the selected range likely encompasses the true number of clusters and is therefore reasonable.

As shown in Figure 9, we set different standard deviation thresholds for defining classification boundaries, ranging from 1 to 5 standard deviations. We also report the performance of GEM on mobile, desktop, and web platforms under these settings. The results show that when using 2 standard deviations as the classification boundary, the accuracy reaches its highest on all platforms. However, if the classification boundaries are further widened, several other metrics decline significantly. This is because broader classification boundaries lead to more OOD samples being misclassified as ID samples. Therefore, using 3 standard deviations as the classification boundary is a reasonable choice.

H OOD Scenario Examples

In this section, we show OOD scenario examples classified by internalization-OOD and extrapolation-OOD scenarios in detail.

Algorithm 1: GEM Algorithm

```
Require : GUI agent $\mathcal{F}$, encoder layer $l_e$, ID dataset $\mathcal{D}_{\text{ID}} = \{(s_i, x_i)\}_{i=1}^{k}$
Ensure  : OOD detection function $f_{\text{OOD}}$
$\mathcal{D}_{\text{embedding}} \leftarrow \emptyset$
for $i \leftarrow 1$ to $k$ do
    $e_i \leftarrow l_e(s_i, x_i)$
    $\mathcal{D}_{\text{embedding}} \leftarrow \mathcal{D}_{\text{embedding}} \cup \{e_i\}$
end
$\mu \leftarrow \frac{1}{k} \sum_{i=1}^{k} e_i$
$\mathcal{D}_{\text{distance}} \leftarrow \emptyset$
for $i \leftarrow 1$ to $k$ do
    $d_i \leftarrow \|e_i - \mu\|_2$
    $\mathcal{D}_{\text{distance}} \leftarrow \mathcal{D}_{\text{distance}} \cup \{d_i\}$
end
for $m \in \{1, \ldots, M\}$ do
    Initialize GMM parameters $\{\pi_j, \mu_j, \sigma_j^2\}_{j=1}^{m}$
    repeat
        for $i \leftarrow 1$ to $k$ do
            for $j \leftarrow 1$ to $m$ do
                $\gamma_{ij} \leftarrow \frac{\pi_j \mathcal{N}(d_i|\mu_j, \sigma_j^2)}{\sum_{l=1}^{m} \pi_l \mathcal{N}(d_i|\mu_l, \sigma_l^2)}$
            end
        end
        for $j \leftarrow 1$ to $m$ do
            $\pi_j \leftarrow \frac{1}{k} \sum_{i=1}^{k} \gamma_{ij}$
            $\mu_j \leftarrow \frac{\sum_{i=1}^{k} \gamma_{ij} d_i}{\sum_{i=1}^{k} \gamma_{ij}}$
            $\sigma_j^2 \leftarrow \frac{\sum_{i=1}^{k} \gamma_{ij} (d_i - \mu_j)^2}{\sum_{i=1}^{k} \gamma_{ij}}$
        end
    until convergence
    $\log \mathcal{L}_m \leftarrow 0$
    for $i \leftarrow 1$ to $k$ do
        $\ell_i \leftarrow \sum_{j=1}^{m} \pi_j \cdot \mathcal{N}(d_i \mid \mu_j, \sigma_j^2)$
        $\log \mathcal{L}_m \leftarrow \log \mathcal{L}_m + \log(\ell_i)$
    end
    BIC$(m) \leftarrow -2 \log \mathcal{L}_m + m \log k$
end
$m^* \leftarrow \arg\min_m$ BIC$(m)$
Fit final GMM with $m^*$ components: $\{(\pi_j, \mu_j, \sigma_j)\}_{j=1}^{m^*}$
for $j \leftarrow 1$ to $m^*$ do
    Define ID interval $I_j = [\mu_j - 3\sigma_j,\ \mu_j + 3\sigma_j]$
end
Define $f_{\text{OOD}}(s, x)$ as:
    $$f_{\text{OOD}}(s, x) = \begin{cases} 0, & \text{if } \|l_e(s, x) - \mu\|_2 \in \bigcup_{j=1}^{m^*} I_j \\ 1, & \text{otherwise} \end{cases}$$
return $f_{OOD}$
```

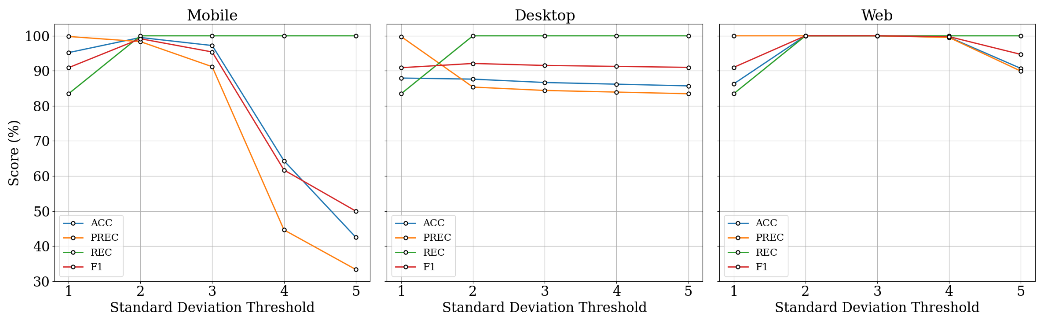

Figure 9: Ablation study on the number of standard deviations as the classification threshold.

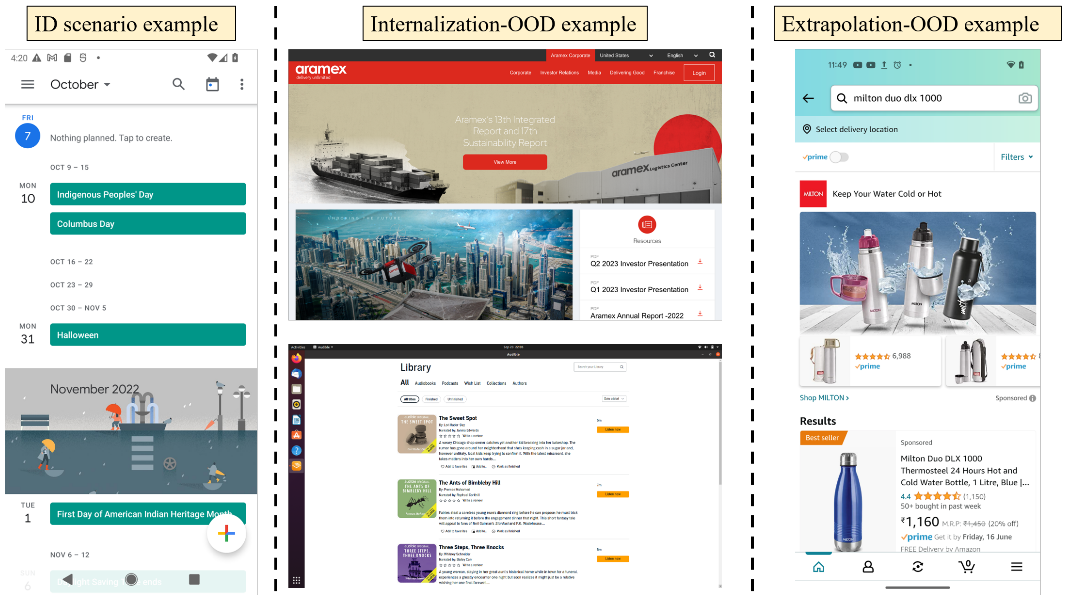
ID scenario example
Internalization-OOD example
Extrapolation-OOD example
October
Nothing planned. Tap to create.
Indigenous Peoples' Day
Columbus Day
Halloween
November 2022
First Day of American Indian Heritage Month
aramex
Q2 2023 Investor Presentation
Q1 2023 Investor Presentation
Aramex Annual Report -2022
Library
The Sweet Spot
The Ants of Bimbleby Hill
Three Steps, Three Knocks
milton duo dlx 1000
Select delivery location
Keep Your Water Cold or Hot
Results
Best seller
Milton Duo DLX 1000 Thermosteel 24 Hours Hot and Cold Water Bottle, 1 Litre, Blue |...

Figure 10: The examples of internalization-OOD and extrapolation-OOD scenarios.

As shown in Figure 10, for a GUI agent trained on the ID dataset AITZ, it has only acquired knowledge related to operating the smartphone platform. Therefore, for this GUI agent, tasks involving the computer and web browser platforms fall under internalization-OOD scenarios. Since the AITZ dataset does not contain operational knowledge about the Amazon application, any task involving Amazon would constitute an extrapolation-OOD scenario for this GUI agent. In these OOD scenarios, the agent's behavior is less reliable and may pose unnecessary risks.