StegoAgent: A Generative Steganography Framework Based on GUI Agents

SHEN Qiuhong, YANG Zijin, JIANG Jun,
ZHANG Weiming, CHEN Kejiang

(University of Science and Technology of China, Hefei 230000, China)

DOI: 10.12142/ZTECOM.202503006

https://kns.cnki.net/kcms/detail/34.1294.TN.20250901.1713.002.html, published online September 2, 2025

Manuscript received: 2025-06-23

Abstract: Steganography is a technology that discreetly embeds secret information into the redundant space of a carrier, enabling covert communication. As generative models continue to advance, steganography has evolved from traditional modification-based methods to generative steganography, which includes generative linguistic and image based forms. However, while large model agents are rapidly emerging, no method has exploited the stable redundant space in their action processes. Inspired by this insightful observation, we propose a steganographic method leveraging large model agents, employing their actions to conceal secret messages. In this paper, we introduce StegoAgent, a generative steganography framework based on graphical user interface (GUI) agents, which effectively demonstrates the remarkable potential and effectiveness of large model agent-based steganographic methods.

Keywords: generative steganography; GUI agent; action

Citation (Format 1): SHEN Q H, YANG Z J, JIANG J, et al. StegoAgent: a generative steganography framework based on GUI agents [J]. ZTE Communications, 2025, 23(3): 48 - 58. DOI: 10.12142/ZTECOM.202503006

Citation (Format 2): Q. H. Shen, Z. J. Yang, J. Jiang, et al., "StegoAgent: a generative steganography framework based on GUI agents," *ZTE Communications*, vol. 23, no. 3, pp. 48 - 58, Sept. 2025. doi: 10.12142/ZTECOM.202503006.

1 Introduction

Steganography[1-2] is a covert communication technology designed to hide the suspicious act of transmitting ciphertext over public channels. It leverages the redundancy of innocent-looking cover objects, embedding secret messages by making plausible modifications. Compared with traditional encryption techniques, steganography effectively avoids transmitting obvious encrypted data (e.g., random bit streams) over public channels. As a result, it reduces the risk of attracting the attention of adversaries and ensures that communication remains secure from interception or tampering. Therefore, steganography can play an important role in ensuring the secure transmission of confidential information. Moreover, it helps mitigate some of the security risks associated with encryption technologies.

Traditional steganography embeds secret messages into covers by modifying their inherent characteristics. This approach allows for the efficient embedding of large data volumes while maintaining high anti-steganalysis performance. With the advancement of deep learning, deep-learning based steganalysis techniques are rapidly developed[3-4], which are capable of extracting steganographic features from stego-covers to detect the presence of embedded secret messages. This development significantly undermines the security of steganographic systems.

To counter steganalysis techniques, traditional steganography has evolved into generative steganography[5]. Generative steganography integrates the embedding process with model generation, achieving stronger security. One of the most notable directions is the combination of generative models and provably secure steganography, which has led to the emergence of generative provably secure steganography, generative provably secure audio steganography, and generative provably secure image steganography. These advances have brought new breakthroughs to the field of steganography, fully proving that steganography is a companion technology.

Over the past two years, the development of large models has evolved from understanding content to generating it, and has increasingly approached human-like intelligent agent technologies—particularly graphical user interface (GUI) agents. GUI agents are intelligent agents that operate within GUI environments, leveraging large language models (LLMs) as their core inference and cognitive engines to generate, plan, and execute actions flexibly and adaptively[6]. However, no steganographic schemes have yet been designed using large-model agents. We observe that the actions of large-model agents have redundant space. Inspired by this observation, we

This work was supported in part by the National Natural Science Foundation of China under Grant Nos. 62472398 and U2336206.
The corresponding author is CHEN Kejiang.

propose StegoAgent, a steganographic agent based on large models, and implement an instance using GUI agents.

The remainder of this paper is organized as follows. Section 2 reviews fundamental concepts and related work in steganography and GUI agents. Section 3 details the architecture and implementation of our StegoAgent framework. Section 4 evaluates the performance of the system through comprehensive experiments. Section 5 concludes with key findings and potential extensions.

2 Related work

2.1 Steganography

Steganography conceals secret information within multimedia covers, such as images[7-9] and videos[10-12]. Traditional steganographic methods are predominantly based on the distortion minimization framework[13], which formalizes the steganography problem as a source coding problem with fidelity constraints. Under the premise of minimizing distortion, steganographic messages are embedded using generic steganographic codes, such as Syndrome Trellis Codes (STCs)[13] and Steganographic Polar Codes (SPCs)[14]. However, these methods modify the distribution of the cover data, resulting in inconsistencies between the distributions of stego data and cover data. This discrepancy makes the steganographic activity susceptible to detection by steganalysis techniques.

With the rapid development of generative models[15-16], an increasing amount of generated data has emerged on social media platforms, providing a novel data ecosystem for steganography. As a result, researchers have shifted their focus towards generative steganography[17-22]. Based on whether the generative model can provide an explicit probability distribution, generative steganography can be categorized into two types. One type[18-19, 22] utilizes the explicit probability distribution of the generated data for message embedding. Under the constraint of finite entropy, it aims to maximize the entropy utilization rate to embed more messages. The other type[17, 20-21], although unable to use an explicit data distribution, can couple the secret information with the initial standard Gaussian distribution of the model generation process, exploring safer and more robust coupling methods.

Previous generative steganographic methods were limited to generating content to convey information. With the rapid development of large model agents, new tools have been introduced to the field of steganography. In this paper, we explore a novel approach to steganography by leveraging the action expressions of intelligent agents to convey secret information. Specifically, we adaptively embed secret messages into the action coordinates of the action flow by leveraging normalized entropy. The interaction process of the GUI agent is then transmitted from the sender to the receiver via screen recording or screen-sharing techniques. Upon receiving the recorded interaction, the receiver reconstructs both the action coordinates and the action stream, from which the embedded secret message is subsequently extracted.

2.2 GUI Agent

GUI automation has a long history and wide application in industry, especially in GUI testing[23-24] and robotic process automation (RPA)[25] for task automation. The rapid development of LLMs has accelerated advancements in GUI automation, enabling more intelligent and efficient interaction with GUIs.

Most early GUI agents focused exclusively on web GUI scenarios, directly perceiving the environment through HyperText Markup Language (HTML) code[26-28]. With the emergence of multimodal LLMs (MLLMs), GUI agents have started to incorporate multiple modalities for environmental perception[29-33], thereby expanding their applicability to both mobile GUIs[34-35] and desktop GUIs[36-37]. Furthermore, cross-platform GUI agents[38-40] have emerged as general-purpose tools capable of interacting with diverse environments, spanning desktop and mobile interfaces to more complex software ecosystems. For example, AutoGLM[41] bridges the gap between web browsing and Android control by integrating large multimodal models for seamless GUI interactions across platforms.

From a mathematical perspective, the workflow of a GUI agent can be formally modeled as follows. Given a GUI interface S (e.g., an online shopping platform) and a user instruction T (e.g., please help me buy a book), the agent computes an executable action sequence $\mathcal{A} = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \cdots, \boldsymbol{a}_n\}$ and interacts with the environment through these actions to fulfill the user instruction. At each time step t, the GUI agent perceives the current environmental state s_t from the screenshot, and then retrieves the sequence of previously executed actions $\{\boldsymbol{a}_1, \boldsymbol{a}_2, \cdots, \boldsymbol{a}_{t-1}\}$ as short-term memory to predict the next action $\boldsymbol{a}_t$.

$$\boldsymbol{a}_t = f_\Theta\left(T, s_t, \{\boldsymbol{a}_1, \boldsymbol{a}_2, \cdots, \boldsymbol{a}_{t-1}\}\right) \tag{1},$$

where f_Θ is the LLM with parameters Θ. Finally, the GUI agent simulates user behavior by executing the generated action $\boldsymbol{a}_t$ through GUI interaction, which leads to the following state transition:

$$S_{t+1} = S\left(\boldsymbol{a}_t\right) \tag{2}.$$

The GUI agent will iteratively repeat the aforementioned steps until the user-given task is completed. There are three crucial and consecutive processes for GUI agents to fulfill user commands[42]:

1) Perception: This requires the GUI agent to maintain precise perceptual awareness of user instructions (T), environmental states (s_t), and the history of executed actions ($\{\boldsymbol{a}_1, \boldsymbol{a}_2, \cdots, \boldsymbol{a}_{t-1}\}$). Accurate state perception enables the GUI

agent to perform efficient action inference.

2) Action reasoning: The GUI agent predicts the appropriate next action based on the information perceived in the preceding step. We refer to the textual reasoning outputs of the GUI agent as the action flow.

3) Execution: The final step involves executing the generated actions and interacting with the GUI interface. The internal executor of the GUI agent translates the planned action sequence into executable commands, effectively emulating patterns of human-GUI interaction.

3 Methods

As mentioned above, the existing generative steganography technology faces high risks of exposure and low quality of stego-covers due to the direct transmission of generated content.

3.1 Application Scenarios

StegoAgent can be applied in two distinct scenarios: the conventional cover transmission scenario and the real-time communication scenario.

1) Cover transmission scenario. In Fig. 1a, the scenario involves a sender Alice, a receiver Bob, and a supervisor Eve. Eve manages a public video platform with user-provided content. Eve conducts pre-publication reviews on all videos. Alice and Bob hide among regular users of the platform to exchange secret messages. Alice acts as a normal video poster, while Bob poses as an ordinary viewer. Most of the videos that Alice publishes are ordinary screen recordings. However, at prearranged times agreed upon with Bob, she posts a video containing hidden messages. Bob then downloads the video and extracts the secret messages. Upon detecting suspicious activity by Alice, Eve will ban her account, thereby disrupting the covert communication channel between Alice and Bob.

2) Real-time communication scenario. As shown in Fig. 1b, the scenario also involves the sender Alice, the receiver Bob, and the supervisor Eve. Eve manages a live-streaming platform that allows registered users to initiate their own live-streaming rooms. Eve periodically inspects the content of live-streaming rooms by randomly entering channels to monitor for suspicious content. Through her authenticated live-streaming platform, Alice publicly displays the execution process of the GUI agent. Alice and Bob agree on a secret signal to initiate secret message transmission. For example, when Alice starts interacting with the audience, she replaces the standard GUI agent with StegoAgent. When Bob recognizes the secret signal, he begins recording the execution process of StegoAgent and decoding the secret message until he receives the stop signal. In this scenario, Alice embeds secret messages while Bob simultaneously extracts them, enabling real-time covert communication.

In both scenarios, the GUI agent model that Alice uses is a proprietary, fine-tuned model that is not publicly accessible. Consequently, the attacker can only access the video recordings or live streaming content transmitted over public channels. In the absence of the GUI agent model, the attacker may at best reconstruct the cursor coordinates from the video or stream, but cannot recover the complete original action flow. Furthermore, in autoregressive models, previously generated tokens directly influence the distribution of subsequent tokens. Due to this sequential dependency, the attacker who lacks access to the complete action sequence is fundamentally unable to extract the embedded secret message.

Coordinate steganography exploits the internal redundancy present in GUI element positioning. Consequently, the observed coordinates are influenced not only by the spatial layout of UI elements but also by the embedding algorithm. The subtle statistical deviations introduced by the steganographic process are masked by the variations in GUI element positioning caused by model fine-tuning. As a result, an attacker would find it extremely difficult to train a reliable classifier based on the reconstructed coordinates for the detection of steganographic activities. Under this realistic adversarial scenario, the embedding coordinates remain statistically indistinguishable from normal, benign coordinates.

3.2 StegoAgent Framework

To address the limitations of current generative steganography, we propose a new framework that utilizes natural covers. This framework requires establishing an invertible mapping between the generated content and these covers.

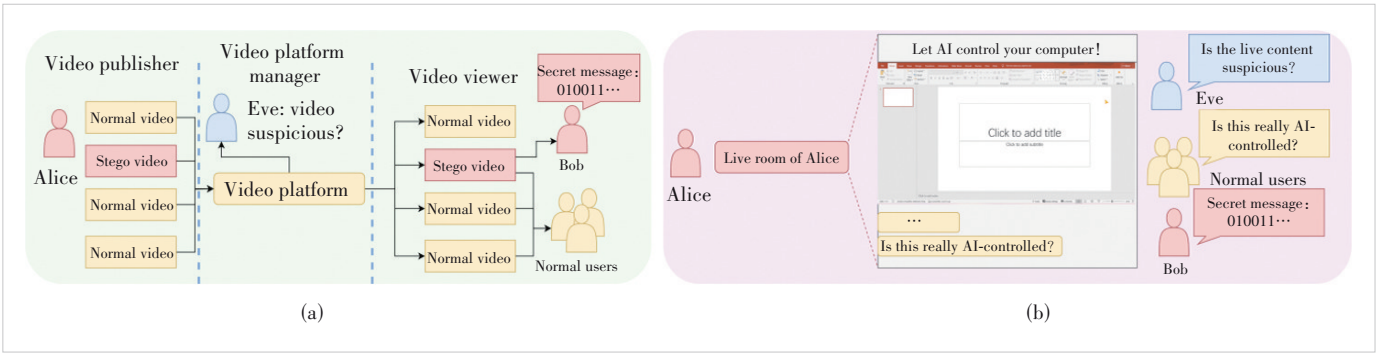


Figure 1. Overview of two application scenarios: (a) conventional cover transmission and (b) real-time communication

As previously discussed, modifying the action flow of the GUI agent can directly alter its execution behavior, thereby establishing a unidirectional mapping from the generated content to natural covers. The action flow of the GUI agent is a formal representation of actions, which can be considered equivalent to the action vector $\boldsymbol{a}_t$ at the corresponding timestep t. During embedding, given a secret message m and an embedding algorithm ϕ, the action containing the secret message $\boldsymbol{a}_t'$ can be expressed as:

$$\boldsymbol{a}_t' = \phi\left(f_\Theta\left(T, s_t, \{\boldsymbol{a}_1', \boldsymbol{a}_2', \cdots, \boldsymbol{a}_{t-1}'\}\right), m\right) \tag{3}$$

With screen recording or screen sharing technology, each executed action can be captured as an action recording $\boldsymbol{p}_{\boldsymbol{a}_t'}$, and all such recordings collectively form the corresponding action recording set $\mathcal{P}'$:

$$\begin{aligned} &\boldsymbol{p}_{\boldsymbol{a}_t'} = \{s_t, s_{t+1}\}, \\ &\mathcal{P}' = \left\{\boldsymbol{p}_{\boldsymbol{a}_1'}, \boldsymbol{p}_{\boldsymbol{a}_2'}, \cdots, \boldsymbol{p}_{\boldsymbol{a}_t'}\right\} = \left\{s_1, s_2, \cdots, s_t, s_{t+1}\right\} \end{aligned} \tag{4}$$

Since screen recording or screen sharing technology can only capture on-screen information, the action recording $\boldsymbol{p}_{\boldsymbol{a}_t'}$ is defined as a tuple consisting of the pre-action state s_t (the screen capture before the action) and the post-action state s_{t+1} (the screen capture after the action). Due to the inherent computational latency in both action reasoning and execution, there exists a measurable delay around each action. This temporal characteristic enables the reliable identification of action boundaries in video recordings and real-time screen streams, thereby allowing for the precise extraction of the corresponding environmental states s_t and s_{t+1}.

During extraction, the receiver reconstructs the corresponding action sequence $\mathcal{A} = \{\boldsymbol{a}_1', \boldsymbol{a}_2', \cdots, \boldsymbol{a}_t'\}$ from the recorded set $\mathcal{P}'$, and then applies the extraction algorithm ψ to recover the secret message:

$$m = \psi\left(f_\Theta\left(T, s_t, \{\boldsymbol{a}_1', \boldsymbol{a}_2', \cdots, \boldsymbol{a}_{t-1}'\}\right), \boldsymbol{a}_t'\right) \tag{5}$$

How can the corresponding action set be reconstructed from $\mathcal{P}'$? We begin by considering the reconstruction of a single-step action, specifically reconstructing $\boldsymbol{a}_t'$ using corresponding recording $\boldsymbol{p}_{\boldsymbol{a}_t'}$. Each action corresponds to a specific application or system event within theenvironment and can be formally represented as a triple:

$$\boldsymbol{a}_t = (\eta, \omega, \nu) \tag{6}$$

In Eq. (6), η represents a target position (e.g., [0.50, 0.20]) as a pixel coordinate on the screen, denoting the position where "Click", "Type", or "Select" operation should be executed; $\omega \in \mathcal{O}$ specifies the intended operation type (e. g., "Click"); ν provides any additional value required for the action (e. g., the type content "hello"). The set $\mathcal{O}$ encompasses all allowable operations within the environment S and is always explicitly defined in the system prompt.

As shown in Eq. 3, $\boldsymbol{a}_t'$ is determined by the embedding algorithm ϕ which in turn influences the performance of the GUI agent. To ensure steganographic security, it is desirable for the embedding of secret messages to minimally affect the performance of the GUI agent. An intuitive approach is to embed the secret message only in those attributes of (η, ω, ν) that have a minimal impact on the performance of the GUI agent, specifically those with higher redundancy. ω is defined within a finite space and exhibits relatively low redundancy. As an auxiliary value, ν exhibits greater redundancy. However, even the most common form of ν (the input content required for "Type" actions) is difficult to reconstruct from a screenshot. η contains relatively high redundancy due to the inherent redundancy present in GUI interface elements. Moreover, it can be bound to cursor actions, allowing its value to be indirectly reconstructed through cursor behavior. In this case, ω and ν can be directly generated from the environmental information s_t in the action recording:

$$\begin{aligned} &\boldsymbol{a}_t = f_\Theta\left(T, s_t, \{\boldsymbol{a}_1', \boldsymbol{a}_2', \cdots, \boldsymbol{a}_{t-1}'\}\right) = (\eta', \omega, \nu), \\ &\eta = g\left(s_{t+1}\right) \end{aligned} \tag{7}$$

where g is the position predictor used to reconstruct η.

After the receiver obtains the action recording set $\mathcal{P}'$, it is straightforward to reconstruct $\boldsymbol{a}_1'$ using s_1, s_2, T, and then recover $\boldsymbol{a}_2'$ based on $\boldsymbol{a}_1'$, s_3, etc. However, this approach has a critical limitation: if the reconstruction fails at any step, all subsequent actions will also fail in reconstruction, resulting in a complete failure of the extraction process. Such a consequence is unacceptable in practical applications. To address this limitation, we observe that certain GUI agents decompose action reasoning into two distinct stages. In the first stage, the overall task T is decomposed into a subtask T_t; in the second stage, the corresponding actions $\boldsymbol{a}_t$ are inferred directly from each T_t, without relying on context. This approach effectively mitigates the aforementioned limitation.

The improved process for embedding secret messages is outlined as follows:

$$\begin{aligned} &T_t = f_{\Theta 1}\left(T, s_t, \{T_1, T_2, \cdots, T_{t-1}\}\right), \\ &\boldsymbol{a}_t' = \phi\left(f_{\Theta 2}\left(T_t, s_t\right), m\right), \\ &\boldsymbol{p}_{\boldsymbol{a}_t'} = \{s_t, s_{t+1}\}, \\ &\mathcal{P}' = \left\{\boldsymbol{p}_{\boldsymbol{a}_1'}, \boldsymbol{p}_{\boldsymbol{a}_2'}, \cdots, \boldsymbol{p}_{\boldsymbol{a}_t'}\right\} = \left\{s_1, s_2, \cdots, s_t, s_{t+1}\right\} \end{aligned} \tag{8}$$

where $f_{\Theta 1}, f_{\Theta 2}$ are the LLMs with parameters $\Theta 1, \Theta 2$. The improved procedure for extracting secret messages is outlined as follows.

$$
\begin{aligned}
T_t &= f_{\Theta 1}\left(T, s_t, \{T_1, T_2, \cdots, T_{t-1}\}\right), \\
\boldsymbol{a}_t &= f_{\Theta 2}\left(T_t, s_t\right) = \left(\eta, \omega, \nu\right), \\
\eta' &= g\left(s_{t+1}\right), \\
\boldsymbol{a}_t' &= \left(\eta', \omega, \nu\right), \\
m &= \psi\left(f_{\Theta 2}\left(T_t, s_t\right), \boldsymbol{a}_t'\right)
\end{aligned} \tag{9}
$$

After the receiver obtains the recorded action sequence $\mathcal{P}'$, they can use a model with parameters $\Theta 1$ to reconstruct the subtask T_t at each step t. Based on this formulation, the original action stream $\boldsymbol{a}_t$ can be reconstructed using T_t and s_t. The cursor position η' is predicted via the position predictor, resulting in the stego text $\boldsymbol{a}_t'$. Applying steganographic extraction techniques then allows the embedded secret message m to be recovered from the stego text $\boldsymbol{a}_t'$.

3.3 StegoAgent Instance

To evaluate the effectiveness of the StegoAgent, we implement the method using Qwen2.5VL-7B (Qwen) [43] and ShowUI[40]. The core model of the GUI agent is based on the ShowUI framework. Additionally, we employ the deployment software of ShowUI, "Computer Use OOTB", to integrate the various modules of the GUI agent. As illustrated in Fig. 2, the GUI agent utilizes Qwen as the planning model, responsible for decomposing the overall task T into an executable subtask T_t. The ShowUI framework functions as the reasoning model, directly inferring the corresponding action $\boldsymbol{a}_t$ from each subtask T_t. First, Qwen generates a subtask based on the task history and user instructions. Next, ShowUI infers the corresponding action sequence in accordance with the subtask. Finally, the executor performs the designated action.

Fig. 3 illustrates the implementation framework of the StegoAgent. Since the steganography method is a generative steganography based on predicted probability distributions and allows message extraction token by token, we merge the steps of reconstructing $\boldsymbol{a}_t'$ and then extracting m in the extraction algorithm:

$$
\begin{aligned}
T_t &= f_{\Theta 1}\left(T, s_t, \{T_1, T_2, \cdots, T_{t-1}\}\right), \\
m &= \psi\left(f_{\Theta 2}\left(T_t, s_t, \right), g\left(s_{t+1}\right)\right)
\end{aligned} \tag{10}
$$

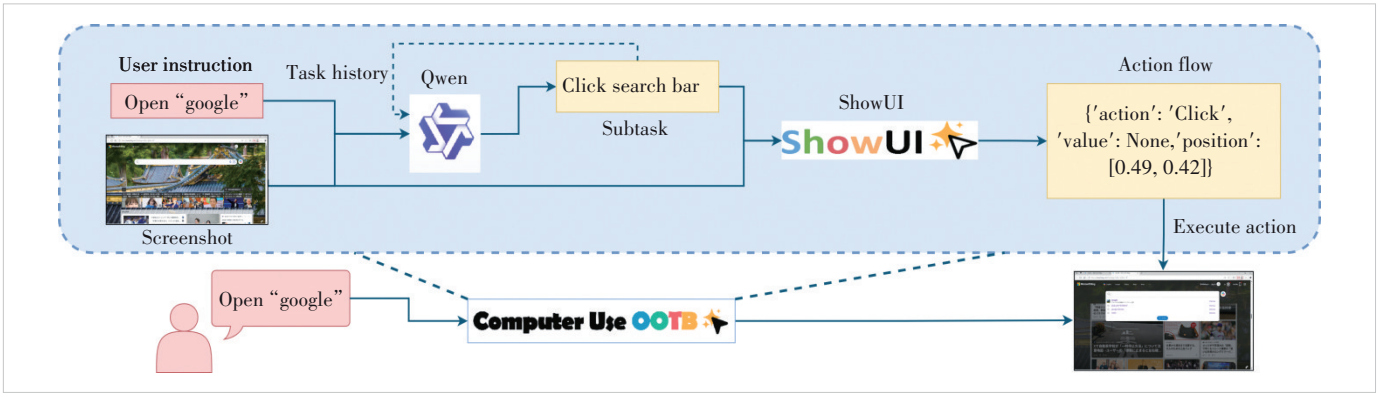


Figure 2. Graphical user interface agent workflow of StegoAgent

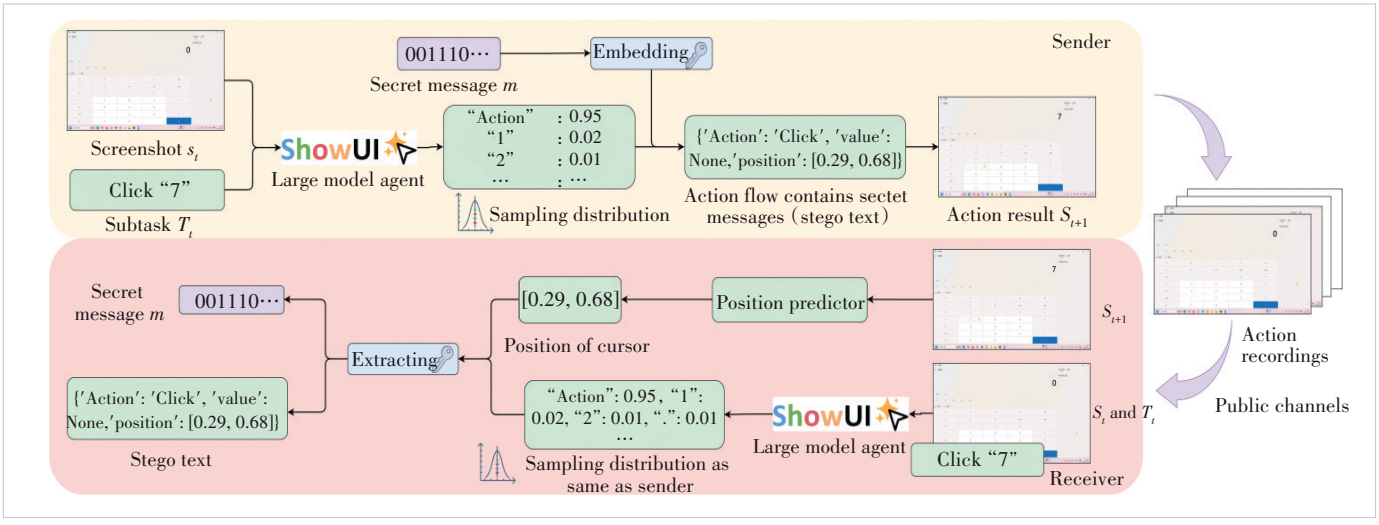


Figure 3. Framework of StegoAgent

Specifically, during the embedding process, the user instruction T and the screenshot are fed into ShowUI, which subsequently predicts a probability distribution over possible actions. By using the embedding algorithm, the secret message is embedded into the segment of the action flow corresponding to action positions. The resulting modified action $\boldsymbol{a}_t'$ is then executed by ShowUI to complete the task. The sender transmits the action recordings of StegoAgent $\mathcal{P}'$ to the receiver. Upon receiving the action recordings, the receiver reconstructs the sequence of screenshots identical to those on the sender side. A position predictor is applied to recover the cursor coordinates (i. e., the action position), which is subsequently transformed into a string formatted according to predefined specifications, followed by tokenization to extract the corresponding tokens. Given a pre-agreed user instruction T, Qwen decomposes T into a subtask T_t. Subsequently, ShowUI predicts the same probability distribution as that of the sender. During the token sampling process, we first determine whether the current token corresponds to an action position. If not, the token is sampled directly. If it does, a coordinate reconstruction token is retrieved, and the extraction function ψ is invoked to extract the secret message.

3.4 Embedding and Extraction

To minimize the impact on the performance of the GUI agent, we use normalized entropy to adaptively embed secret messages. As shown in Algorithm 1, given a sorted token probability sequence *probs*, along with a base b and a threshold ϵ, we first calculate the normalized entropy over the top 2^b tokens. If this entropy value is greater than ϵ, we proceed to embed b bits of secret information. Otherwise, the base b is decremented, and the normalized entropy is recalculated using the updated top $b-1$ tokens. This adaptive procedure iterates until either a suitable embedding capacity is found or b reaches zero, in which case no message is embedded for that token.

Algorithm 1. Adaptive Steganographic Embedding via Normalized Entropy

procedure: Embed message ($probs$, b_{init}, ϵ, m)
  input: Sorted token sequence *probs*, initial base b_{init}, threshold ϵ, and secret message m
  output: Token with embedded message t_m or no embedding
  $b \leftarrow b_{\text{init}}$
  while $b \geqslant 0$ **do**
    Select top 2^b tokens from *probs*
    Compute normalized entropy of selected tokens
    if entropy $> \epsilon$ **then**
      $m_b \leftarrow$ First b bits of m
      $d \leftarrow \text{binary\_to\_decimal}(m_b)$
      $t_m \leftarrow probs[d]$
      return t_m
    else
      $b \leftarrow b - 1$
    end if
  end while
  if $b < 0$ **then**
    return $probs[0]$
  end if
end procedure

In Algorithm 2, during extraction we are given a sorted token sequence *probs*, a base b, a threshold ϵ, and the stego-token t_s. We first compute the normalized entropy over the top 2^b tokens. If the entropy exceeds the threshold ϵ, we extract the index of the stego-token t_s within the top 2^b tokens and convert it into a bitstream, which constitutes the secret message. If the normalized entropy is below ϵ, we reduce the base b and recompute the normalized entropy for comparison. This process continues until b reaches zero, at which point we conclude that the token does not contain any embedded secret message.

3.5 Position Predictor

To reconstruct the cursor's relative position in the screenshot, we propose a position predictor. Compared with other elements in the GUI interface, the cursor typically has a consistent appearance; for example, the most common standard cursor is a white arrow with a black border. Based on this characteristic, we design a position estimation approach that utilizes both the color and contour information of the cursor. Specifically, the process begins by isolating the regions in the screenshot that exhibit color similarity to the cursor. The entire screenshot is then binarized: pixels with colors close to those of the cursor are assigned a white value, while all others are assigned a black value. Next, all contours are extracted from the binarized screenshot, and the contour that most closely matches the cursor's expected shape is selected as the estimated cursor position.

Algorithm 2. Steganographic Message Extraction via Normalized Entropy

procedure: Extract message ($probs$, t_s, b_{init}, ϵ)
  input: Sorted token sequence *probs*, stego-token t_s, initial base b_{init}, threshold ϵ
  output: Extracted bitstring m or None if no message is embedded
  $b \leftarrow b_{\text{init}}$
  while $b \geqslant 0$ **do**
    Select top 2^b tokens from probs
    Compute normalized entropy of selected tokens
    if entropy $> \epsilon$ **then**
      Find index i of t_s within the selected 2^b tokens
      Convert i to b-bit binary string m_b
      return m_b

```
    else
        b ← b − 1
    end if
  end while
  return None (no message embedded)
end procedure
```

Fig. 4 illustrates an example of the position predictor, demonstrating its capability to accurately detect the cursor and determine its position within the screenshot. It is worth noting that this example does not use the most classic cursor. The white value of the classic cursor is commonly found throughout various GUI interfaces, making accurate cursor position reconstruction particularly challenging. Therefore, a more distinctive cursor is utilized in this case. With the widespread adoption of internet technologies, cursor personalization has become simple and commonplace. Furthermore, standard cursors vary across different operating systems. Consequently, the use of a distinctive cursor does not undermine the security.

However, this method has a limitation: when the cursor color is similar to the background, it becomes difficult to determine an appropriate color threshold to effectively distinguish the cursor from the background. To address this issue, we incorporate a template matching algorithm to complement the position predictor.

Template matching is a classical image-to-image comparison technique. It works by sliding a template image (i.e., the cursor image) across the target image as a moving window, and computing the similarity score at each position. The location with the highest similarity score is considered the best match. Experimental results show that although the accuracy of template matching is lower than that of the position predictor, it still achieves over 90% accuracy. Therefore, we combine the two algorithms to further improve the overall prediction performance.

Specifically, we set a similarity threshold α. When the similarity score of the best matching contour is below α (note that a lower score indicates a closer match), we accept that contour as the cursor position. If the score is higher than α, we instead use the result from the template matching algorithm to determine the cursor location.

4 Experiments

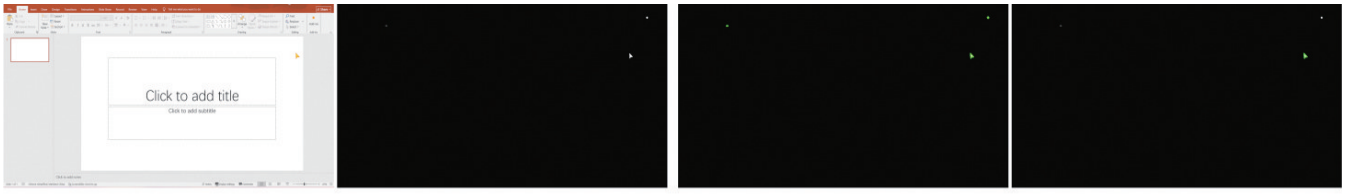
In this section, we conducted experiments to evaluate both the steganographic capabilities and the impact of the proposed method on the GUI Agent.

4.1 Implementation Details

1) Datasets. The performance of the GUI agent is evaluated from two perspectives using the Screenspot[44] and Mind2Web[45] datasets. Screenspot is a zero-shot visual grounding benchmark that includes data from three distinct device types, focusing on the recognition performance of text and widgets. Mind2Web is a web-based dataset with an action space consisting of three distinct actions, designed to assess the overall performance of GUI agents. Additionally, the steganographic capabilities of the GUI agent are also evaluated using these two datasets. Among them, Screenspot includes 1 272 screenshots collected from multiple platforms, while the test set of Mind2Web comprises 9 268 action-context pairs. Since most of the screenshots in both datasets have been cropped, the screenshots exhibit varying sizes and aspect ratios. To ensure uniformity in processing, for any screenshot where the length or width exceeds 2 160 pixels, the dimension exceeding this threshold is resized to 2 160 pixels, maintaining the original aspect ratio. In the extraction accuracy test, we randomly sampled one quarter of the Mind2Web dataset (approximately 2 000 samples) using a random seed of 2 553. Then we discarded samples in which no secret message had been embedded, resulting in a final test set of 1 267 samples. Note that the original screenshots in the datasets do not include a cursor. To test the accuracy of the position predictor, we pasted a cursor at the top-left corner of the annotated UI element regions in the datasets, simulating accurate clicking behavior.

2) Baselines. To evaluate the impact of integrating steganographic algorithms on the performance of the agent model, we adopt ShowUI[40] as the baseline for comparison with StegoAgent. ShowUI, as the base model, employs greedy decoding during sampling to prioritize accuracy. For StegoAgent, the base b is set to 3, the threshold ϵ to 0.96, and the position predictor's threshold α to 0.1.

3) Evaluation metrics. We evaluate the steganography per-



Original screenshot | Binarized screenshot | All contours of binarized screenshot | Most matches contours position: [0.88, 0.27]

Figure 4. Instance of position predictor

formance of the GUI agent using two key aspects: GUI agent ability evaluation and Steganographic performance evaluation.

GUI agent ability evaluation focuses on assessing the agent's performance in grounding and navigation tasks. Grounding capability refers to the ability to identify and locate UI elements in a screenshot, where the agent infers only a coordinate. Navigation capability evaluates the agent's ability to generate a complete action flow, simulating real-world scenarios. To measure grounding performance, we use "Accuracy" as the evaluation metric on the Screenspot[44] dataset, i.e., the recognition rate of texts and widgets. For navigation performance, we use three evaluation metrics on Mind2Web[45] dataset: element identification accuracy (*Ele.Acc*), operation prediction F1 score (*Op. F1*), and step success rate (*Step.SR*). As the focus is on the impact of the proposed steganographic method, all evaluations are conducted in a zero-shot manner without fine-tuning, leading to relatively low accuracy scores.

We evaluate steganographic performance by measuring how integrating steganographic algorithms impacts agent performance. The evaluation considers two aspects: capacity and extraction accuracy. Capacity is quantified by the embedding rate, defined as the average number of bits embedded per generated token. In practice, the steganographic capacity measured in bits per token has limited reference value. Therefore, we measure the capacity in terms of bits per sample. In the grounding test, this represents the average number of bits embedded per coordinate, while in the navigation test, it reflects the average number of bits embedded per action. Extraction accuracy evaluates the ability to retrieve embedded information and includes:

1) Position predictor accuracy. We evaluate position predictor accuracy using the metric "Accuracy", which is defined as:

$$\text{Accuracy} = \frac{\left|\{i|\hat{y}_i = y_i\}\right|}{N}, i = 1,2,\cdots,n \tag{11},$$

where y_i denotes the true position coordinates, and $\hat{y}_i$ denotes the predicted position coordinate for the i-th sample.

2) Overall extraction accuracy. Since the Screenspot[44] dataset differs to some extent from real-world scenarios, overall extraction accuracy is evaluated only on the Mind2Web[45] dataset. We evaluate extraction accuracy using the metric "Bit Accuracy", which is defined as:

$$\text{Bit Accuracy} = \frac{n}{N} \tag{12},$$

where n is the number of correctly extracted bits, and N is the total number of embedded bits.

4.2 Main Performance and Analysis

1) Steganographic extraction accuracy. As shown in Table 1, the prediction accuracy of the position predictor exceeds 97.6%, while that of template matching is only 91.9%. Our proposed prediction method achieves significantly higher accuracy than the traditional template matching algorithm. To address the limitations of the position predictor, we combine the two methods. The combined approach achieves an accuracy of over 99.5% on both datasets, with almost no prediction errors. The StegoAgent achieves a 99.7% secret message extraction accuracy, validating its reliability in retrieving embedded information.

2) Capacity and entropy utilization. Table 2 summarizes the steganographic capacity of StegoAgent. On average, each token supports the embedding of 0.12 bits, while each coordinate provides a total capacity of approximately 1.5 bits. In practical application scenarios, the majority of tokens are not action coordinate tokens. As a result, the steganographic capacity measured in bits per token decreases in the Mind2Web dataset. However, the actual embedding capacity per action remains unchanged. In fact, the effective embedding capacity increases in navigation tasks, yielding an average of about 1.7 bits per action. To maintain behavioral consistency and imperceptibility, we deliberately prioritize stealth over maximizing embedding capacity.

In addition to testing on the dataset, we conducted a small-scale real-world experiment to evaluate StegoAgents steganographic capacity. We selected four websites from the Mind2Web dataset, and for each website, we defined five representative tasks resulting in a total of 20 tasks. StegoAgent was instructed to autonomously control the computer to complete each task, and we recorded two-minute videos for each session to measure the embedding capacity per minute of video. The sample size of the experiment is relatively small, as GUI agent-driven computer control is inherently a high-risk process that necessitates manual oversight. As such, the results are meant to serve as a preliminary reference for steganographic capacity in realistic application settings, rather than a comprehensive evaluation. Across the 20 recorded videos, StegoAgent achieved an average steganographic capacity of approximately 2.1 bits per minute.

The real-world steganographic capacity of StegoAgent is

Table 1. Position prediction accuracy

Dataset	TM	Pos	TM+Pos
Screenspot[44]	0.932	0.999	1
Mind2web[45]	0.919	0.976	0.995

TM: template matching
Pos: position predictor
TM+Pos: combined method

Table 2. Results of capacity evaluation

Dataset	Entropy Bit per Token	Capacity Bit per Token	Capacity Bit per Sample
Screenspot[44]	0.383	0.122	1.553
Mind2web[45]	0.438	0.056	1.716

strongly correlated with the performance of the baseline GUI agent model. This is because current GUI agents operate through a multi-stage pipeline: first reasoning about the action sequence, then parsing the action flow, and finally executing the action. Each stage introduces inevitable latency, resulting in most of the recorded video time being spent waiting for the model to perform reasoning and parse the action sequence. Reducing this latency remains a key research challenge in GUI agent development. We believe that as related technologies advance and execution delays decrease, the steganographic capacity of StegoAgent in real-world scenarios will improve significantly.

3) GUI agent ability evaluation. As shown in Tables 3 and 4, the grounding performance of ShowUI experiences a slight degradation after integrating the steganographic algorithm, with an approximate 0.5% decrease in accuracy. Nevertheless, the overall performance remains reasonably acceptable. In the case of grounding tasks, where the model only generates the coordinates of the associated UI elements, the steganographic algorithm directly alters these values, leading to a decrease in accuracy across all element categories. On average, StegoAgent performs nearly identically to ShowUI, demonstrating that the steganographic mechanism introduces negligible impact. The steganographic algorithm does not alter the intended action at each step, resulting in an action F1 score that remains comparable to that of ShowUI. Although the steganography method directly modifies the action coordinates, the accuracy of element identification shows no significant degradation and in certain tasks, even slight improvements over ShowUI are observed. In terms of per-step success rates, StegoAgent exhibits fluctuations around the performance of ShowUI, indicating comparable overall effectiveness. These results collectively demonstrate that StegoAgent maintains strong behavioral consistency with the baseline model while ensuring secure information transmission.

As illustrated in Fig. 5, the coordinate changes before and after steganography are minimal, with some remaining entirely unchanged. This demonstrates that StegoAgent preserves a high degree of behavioral consistency, thereby enhancing its resistance to detection by third parties.

Table 3. Results of grounding capability evaluation accuracy (%)

Method	Mobile Text	Mobile Icon	Desktop Text	Desktop Icon	Web Text	Web Icon	Avg.
ShowUI[40]	0.791	0.672	0.763	0.614	0.804	0.592	0.706
StegoAgent	0.787	0.681	0.758	0.600	0.804	0.578	0.701

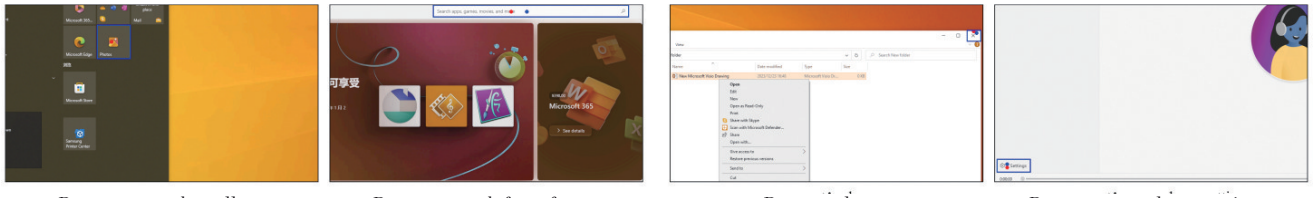
5 Conclusions

We innovatively propose a generative steganographic framework, StegoAgent, using natural media as covers. The core advantages of the StegoAgent lie in its simplicity and efficiency. By requiring only a preshared secret key and a set of instruction prompts, it enables the embedding of secret messages into common media such as natural images and videos. StegoAgent also extends the application scenarios of steganography, enabling real-time transmission of secret messages between the sender and the receiver.

The extraction and embedding processes of StegoAgent are implemented using the lightweight agent model, ShowUI, and the generative steganography method, thereby demonstrating the feasibility of the proposed approach. Furthermore, experiments show that the StegoAgent does not significantly degrade model performance, enabling effective secret message transmission while maintaining the capabilities of the intelligent agent. In addition, we measure the capacity and extraction ac-

Table 4. Results of navigation capability evaluation accuracy (%)

Method	Cross-Task			Cross-Domain			Cross-Website		
	Ele.Acc	Op.F1	Step.SR	Ele.Acc	Op.F1	Step.SR	Ele.Acc	Op.F1	Step.SR
ShowUI[40]	0.214	0.832	0.178	0.248	0.802	0.200	0.224	0.799	0.169
StegoAgent	0.212	0.832	0.179	0.244	0.802	0.196	0.226	0.799	0.170



Prompt: open the gallery | Prompt: search for software | Prompt: close | Prompt: open player settings

Figure 5. Visualization of StegoAgent before and after steganography, where blue bounding boxes delineate the regions of UI elements annotated in the dataset, blue dots represent the coordinates generated by StegoAgent, and red dots indicate the original coordinates

curacy of the StegoAgent and comprehensively evaluate the steganographic performance from multiple perspectives.

References

[1] BARNI M. Steganography in digital media: principles, algorithms, and applications [J]. IEEE signal processing magazine, 2011, 28(5): 142 – 144. DOI: 10.1109/MSP.2011.941841
[2] PEVNÝ T, FRIDRICH J. Benchmarking for steganography [C]//International Workshop on Information Hiding (10th International Workshop). ISIH, 2018. DOI: 10.1007/978-3-540-88961-8_18
[3] REINEL T S, RAÚL R P, GUSTAVO I. Deep learning applied to steganalysis of digital images: a systematic review [J]. IEEE access, 2019, 7: 68970 – 68990
[4] KHEDDAR H, HEMIS M, HIMEUR Y, et al. Deep learning for steganalysis of diverse data types: a review of methods, taxonomy, challenges and future directions [J]. Neurocomputing, 2024, 581: 127528. DOI: 10.1016/j.neucom.2024.127528
[5] LIU J, KE Y, ZHANG Z, et al. Recent advances of image steganography with generative adversarial networks [J]. IEEE access, 2020, 8: 60575 – 60597
[6] ZHANG C Y, HE S L, QIAN J X, et al. Large language model-brained GUI agents: a survey [EB/OL]. (2024-11-27) [2025-06-01]. https://arxiv.org/abs/2411.18279
[7] LIU M L, SONG T T, LUO W Q, et al. Adversarial steganography embedding via stego generation and selection [J]. IEEE transactions on dependable and secure computing, 2023, 20(3): 2375 – 2389. DOI: 10.1109/TDSC.2022.3182041
[8] LI Q, MA B, FU X P, et al. Robust image steganography via color conversion [J]. IEEE transactions on circuits and systems for video technology, 2025, 35(2): 1399 – 1408. DOI: 10.1109/TCSVT.2024.3466961
[9] FAN Z X, CHEN K J, ZENG K, et al. Natias: neuron attribution-based transferable image adversarial steganography [J]. IEEE transactions on information forensics and security, 2024, 19: 6636 – 6649. DOI: 10.1109/TIFS.2024.3421893
[10] LI Z H, JIANG X H, DONG Y, et al. An anti-steganalysis HEVC video steganography with high performance based on CNN and PU partition modes [J]. IEEE transactions on dependable and secure computing, 2023, 20(1): 606 – 619. DOI: 10.1109/TDSC.2022.3140899
[11] HE S H, XU D W, YANG L, et al. Adaptive HEVC video steganography with high performance based on attention-net and PU partition modes [J]. IEEE transactions on multimedia, 2023, 26: 687 – 700. DOI: 10.1109/TMM.2023.3269663
[12] MAO X Y, HU X X, PENG W L, et al. From covert hiding to visual editing: robust generative video steganography [C]//The 32nd ACM International Conference on Multimedia. ACM, 2024: 2757 – 2765. DOI: 10.1145/3664647.3681149
[13] FILLER T, JUDAS J, FRIDRICH J. Minimizing additive distortion in steganography using syndrome-trellis codes [J]. IEEE transactions on information forensics and security, 2011, 6(3): 920 – 935. DOI: 10.1109/TIFS.2011.2134094
[14] LI W X, ZHANG W M, LI L, et al. Designing near-optimal steganographic codes in practice based on polar codes [J]. IEEE transactions on communications, 2020, 68(7): 3948 – 3962. DOI: 10.1109/TCOMM.2020.2982624
[15] GOODFELLOW I J, POUGET-ABADIE J, MIRZA M, et al. Generative adversarial networks [C]//The 28th International Conference on Neural Information Processing Systems. ACM, 2014
[16] ROMBACH R, BLATTMANN A, LORENZ D, et al. High-resolution image synthesis with latent diffusion models [C]//Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 2022: 10674 – 10685. DOI: 10.1109/CVPR52688.2022.01042
[17] PENG F, CHEN G F, LONG M. A robust coverless steganography based on generative adversarial networks and gradient descent approximation [J]. IEEE transactions on circuits and systems for video technology, 2022, 32(9): 5817 – 5829. DOI: 10.1109/TCSVT.2022.3161419
[18] DING J Y, CHEN K J, WANG Y F, et al. Discop: provably secure steganography in practice based on "distribution copies" [C]//IEEE Symposium on Security and Privacy (SP). IEEE, 2023: 2238 – 2255. DOI: 10.1109/SP46215.2023.10179287
[19] WITT D C S, SOKOTA S, KOLTER J Z, et al. Perfectly secure steganography using minimum entropy coupling [C]//The Eleventh International Conference on Learning Representations. ICLR, 2023:1 – 14
[20] YANG Z J, CHEN K J, ZENG K, et al. Provably secure robust image steganography [J]. IEEE transactions on multimedia, 2023, 26: 5040 – 5053. DOI: 10.1109/TMM.2023.3330098
[21] HU X X, LI S, YING Q C, et al. Establishing robust generative image steganography via popular stable diffusion [J]. IEEE transactions on information forensics and security, 2024, 19: 8094 – 8108. DOI: 10.1109/TIFS.2024.3444311
[22] WANG Y F, PEI G, CHEN K J, et al. Sparsamp: efficient provably secure steganography based on sparse sampling [EB/OL]. [2025-06-01]. https://www. usenix. org/system/files/conference/usenixsecurity25/sec25cycle1-prepub-240-wang-yaofei.pdf
[23] LI K L, WU M Q. Effective GUI testing automation: developing an automated GUI testing tool [M]. Hoboken, USA: John Wiley & Sons, 2006
[24] RODRÍGUEZ-VALDÉS O, EJ VOS T, AHOV P, et al. 30 years of automated GUI testing: a bibliometric analysis [C]//The 14th International Conference Quality of Information and Communications Technology. CCIS, 2021: 473 – 488
[25] IVANČIĆ L, SUŠA VUGEC D, BOSILJ VUKŠIĆ V. Robotic process automation: systematic literature review [EB/OL]. [2025-06-01]. https://link.springer.com/content/pdf/10.1007/978-3-030-30429-4_19.pdf
[26] GUR I, FURUTA H, HUANG A V, et al. A real-world webagent with planning, long context understanding, and program synthesis [EB/OL]. (2023-07-24) [2025-06-01]. https://arxiv.org/abs/2307.12856
[27] KIM G, BALDI P, MCALEER S. Language models can solve computer tasks [C]//The 37th International Conference on Neural Information Processing Systems. NIPS, 2023: 39648 – 39677
[28] LO R, SRIDHAR A, XU F, et al. Hierarchical prompting assists large language model on web navigation [C]//Proceedings of Findings of the Association for Computational Linguistics. EMNLP. Association for Computational Linguistics, 2023: 10217 – 10244. DOI: 10.18653/v1/2023.findings-emnlp.685
[29] LAI H Y, LIU X, IONG I L, et al. AutoWebGLM: a large language model-based web navigating agent [C]//The 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. ACM, 2024: 5295 – 5306. DOI: 10.1145/3637528.3671620
[30] AGASHE S, HAN J Z, GAN S Y, et al. Agent S: an open agentic framework that uses computers like a human [C]//The Thirteenth International Conference on Learning Representations. ICLR, 2025
[31] NIU R L, LI J D, WANG S Q, et al. ScreenAgent: a vision language model-driven computer control agent [C]//Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence. International Joint Conferences on Artificial Intelligence Organization, 2024: 6433 – 6441. DOI: 10.24963/ijcai.2024/711
[32] HE H L, YAO W L, MA K X, et al. WebVoyager: building an end-to-end web agent with large multimodal models [C]//Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). ACL, 2024: 6864 – 6890. DOI: 10.18653/v1/2024.acl-long.371
[33] IONG I L, LIU X, CHEN Y X, et al. OpenWebAgent: an open toolkit to enable web agents on large language models [C]//The 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3:

System Demonstrations). ACL, 2024: 72 – 81. DOI: 10.18653/v1/2024.acl-demos.8
[34] WANG B, LI G, LI Y. Enabling conversational interaction with mobile UI using large language models [C]//The 2023 CHI Conference on Human Factors in Computing Systems. ACM, 2023: 1 – 17. DOI: 10.1145/3544548.3580895
[35] ZHANG C, YANG Z, LIU J X, et al. AppAgent: multimodal agents as smartphone users [C]//Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems. ACM, 2025: 1 – 20. DOI: 10.1145/3706598.3713600
[36] ZHANG C Y, LI L Q, HE S L, et al. UFO: a UI-focused agent for windows OS interaction [C]//The 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies. ACL, 2025: 597 – 622. DOI: 10.18653/v1/2025.naacl-long.26
[37] WU Z Y, HAN C C, DING Z C, et al. Os-copilot: towards generalist computer agents with self-improvement. [EB/OL]. (2024-02-12) [2025-06-01]. https://arxiv.org/abs/2402.07456
[38] AGASHE S, WONG K, TU V, et al. Agent s2: a compositional generalist-specialist framework for computer use agents. [EB/OL]. (2025-04-01) [2025-06-01]. https://arxiv.org/abs/2504.00906
[39] WANG Y Q, ZHANG H J, TIAN J Q, et al. Ponder & press: advancing visual GUI agent towards general computer control [C]//Findings of the Association for Computational Linguistics. ACL, 2025: 1461 – 1473
[40] LIN K Q H, LI L J, GAO D F, et al. ShowUI: one vision-language-action model for GUI visual agent [EB/OL]. (2024-11-26) [2025-06-01]. https://arxiv.org/abs/2411.17465
[41] LIU X, QIN B, LIANG D Z, et al. Autoglm: autonomous foundation agents for GUIs [EB/OL]. (2024-10-28) [2025-06-01]. https://arxiv.org/abs/2411.00820
[42] NING L B, LIANG Z R, JIANG Z H, et al. A survey of webagents: towards next-generation AI agents for web automation with large foundation models [C]//The 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining. ACM, 2025: 6140 – 6150
[43] BAI S, CHEN K Q, LIU X J, et al. Qwen2.5-VL technical report [EB/OL]. (2025-02-19) [2025-06-01]. https://arxiv.org/abs/2502.13923
[44] CHENG K Z, SUN Q S, CHU Y G, et al. Seeclick: harnessing GUI grounding for advanced visual GUI agents [C]//The 62nd Annual Meeting of the Association for Computational Linguistics. ACL, 2024: 9313 – 9332. DOI: 10.18653/v1/2024.acl-long.505
[45] DENG X, GU Y, ZHENG B Y, et al. Mind2web: towards a generalist agent for the web [C]//The 37th International Conference on Neural Information Processing Systems. ACM, 2023: 28091 – 28114

Biographies

SHEN Qiuhong received her BS degree from the University of Science and Technology of China (USTC) in 2025. She is currently pursuing her MS degree at USTC. Her research interests include information hiding and multimedia security.

YANG Zijin received his BS degree from the University of Science and Technology of China (USTC) in 2022. He is currently pursuing a PhD degree in engineering at the School of Cyber Science and Technology, USTC. His research interests include information hiding and multimedia security.

JIANG Jun received his BS degree from Shanghai University, China in 2024 and is currently pursuing his MS degree at the University of Science and Technology of China. His research interests include information hiding and model security.

ZHANG Weiming received his MS and PhD degrees from the Zhengzhou Information Science and Technology Institute, China in 2002 and 2005, respectively. Currently, he is a professor at the School of Information Science and Technology, University of Science and Technology of China. His research interests include information hiding and multimedia security.

CHEN Kejiang (chenkj@mail.ustc.edu.cn) received his BS degree from Shanghai University, China and PhD degree from the University of Science and Technology of China (USTC) in 2015 and 2020, respectively. Currently, he is an associate professor at USTC. His research interests include information hiding, image processing, and deep learning.