AccuAnnotate: Scalable Labelling of Graphical User Interfaces and Reinforcement Learning for Vision-Language Model Grounding

NING BAO

Supervisor: Dr. Hazem El-Alfy

This thesis is submitted in partial fulfillment of
the requirements for the degree of
Bachelor of Advanced Computing (Honours)

School of Computer Science
The University of Sydney
Australia

THE UNIVERSITY OF SYDNEY

Abstract

Over the past few years, the rise of Generative AI has pushed Multimodal Large Language Models, especially Vision-Language Models (VLMs), to the forefront of computer-use agent research. This work helps computers understand software screens by automatically labelling interface elements and training AI to click accurately. We study the specific problem of pixel-level grounding on desktop GUIs: given a screenshot and a natural-language instruction, the agent must predict an exact on-screen click point. The problem is important because reliable GUI agents depend on precise targets for usability and safety in real applications. Yet the field lacks abundant, clean, desktop-specific annotations and current datasets under-represent diverse widgets while auto-labels are often imprecise, constraining both supervised learning and RL fine-tuning. This Honours thesis introduces two contributions: (i) ACCUANNOTATE, an automated, scalable pipeline for high-fidelity GUI annotation, and (ii) ACCUANNOTATE-2B, a compact VLM fine-tuned with reinforcement learning for precise desktop grounding.

ACCUANNOTATE couples OmniParser-v2 [1] element discovery with image pre/post-processing, instruction validation, a crop-level prompt builder, and a web workspace for controllable detail and organisation; manual verification over 1,740 tasks yields **98.77%** correct labels. ACCUANNOTATE-2B is trained based on ShowUI-2B and optimises a distance-based reward. Under a deterministic protocol on ScreenSpot-desktop (N=334) [2] and ScreenSpot-Pro (N=1,581) [3], we observe AUC gains of **+5.29%** and **+16.95%**, respectively, with DTB reduced by **9.08%** on ScreenSpot-desktop and slightly on ScreenSpot-Pro, implicating that scalable, high-precision labeling combined with lightweight RL can materially improve pixel-accurate GUI grounding without resorting to massive models.

- The **ShowUI-2B** backbone served as the starting point for fine-tuning; I thank its authors for releasing models and training artefacts that enabled reproducible baselines.
- **OmniParser V2** was integral to my ACCUANNOTATE pipeline for proposing high-precision bounding boxes and screen parses.
- Open-source frameworks and libraries—including *PyTorch*, *Hugging Face Transformers*, *BitsAndBytes*, *TensorBoard*, *tqdm*, *Pillow (PIL)*, and *Flask*.

Family and friends. This work was made possible by those who believed in me. I thank my **parents** for sponsoring my tertiary study and their unwavering support, my **relatives** in Sydney for practical help and encouragement, and my **friends** for motivation and patience throughout long training runs and late-night debugging.

Closing note. Research is collaborative even when a thesis lists a single author. I am indebted to the researchers who shared datasets and code, the open-source communities that build our tools, and the people in my life who provided time, space, and kindness. Thank you.

CONTENTS

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Background

Early approaches to user interface (UI) interaction automation often relied on textual interface structures, such as HTML DOMs or accessibility trees, rather than raw pixels [5–7]. These context–structure–based agents can parse underlying UI metadata, but they struggle with visual elements that are not exposed in text. This limitation motivated visual UI agents that perceive screenshots directly. A persistent challenge is obtaining sufficient, high-quality training data of desktop graphical user interface (GUI) screenshots with aligned actions or descriptions, which is costly and labour-intensive.

Researchers have explored multimodal learning techniques to enhance the understanding of UI screens. *Screen2Words* generates concise text summaries of mobile screens [8]; *Widget Captioning* describes individual components using visual and structural context [9]; models such as *UI-BERT* and *ActionBERT* integrate screenshots with textual metadata to learn joint representations for UI understanding and action prediction [10, 11]. In parallel, the community produced datasets and environments: *Rico* provides ∼66k mobile screens with view hierarchies [12]; simulator suites such as *MiniWob++*, *Mind2Web*, and *OSWorld* enable controlled agent evaluation [13–15]. Whilst valuable, simulators and mobile-focused datasets offer few large-scale desktop screenshots with pixel-precise annotations, which are essential for training vision–language models (VLMs).

To improve grounding, hybrid prompting has emerged as a solution. Set-of-Marks (SoM) overlays labelled marks so a model can refer to region IDs instead of emitting raw coordinates [16–18]. More recently, *OmniParser* introduced a purely vision-based screen parsing pipeline that detects UI elements and captions them, supplying structured boxes and labels without DOM access. With OmniParser, GPT-class VLMs improve performance on ScreenSpot-style benchmarks [1–3]. Despite these advances, creating large, accurate, desktop-focused annotation sets remains a bottleneck. *ShowUI* demonstrated

that carefully chosen GUI data can unlock strong zero-shot grounding ability for compact models [19], yet assembling such data is labour-intensive, and complementary reinforcement learning (RL) approaches reduce the need for labelled data but still benefit from reliable supervision [20].

1.2 Our Work

We address both the data and the model sides. First, we build ACCUANNOTATE, an end-to-end annotation system that converts raw desktop screenshots into pixel-accurate bounding boxes and concise instructions with high agreement to a human rater (98.77% on N=1,740 tasks; see Chapter 4). Second, we introduce ACCUANNOTATE-2B, a compact vision–language model for desktop GUI grounding. ACCUANNOTATE-2B starts from the *ShowUI-2B* backbone [19] and is fine-tuned with a lightweight reinforcement-learning approach to address the gap: (i) a *distance-aware reward* that transforms binary inside-box success into a dense shaping signal toward element centres, (ii) an EMA advantage baseline and moderate entropy to stabilise updates, (iii) a tightening success-radius schedule (τ).

1.3 Research Question

> **How can we automatically produce high-fidelity, pixel-accurate annotations for desktop GUI grounding, and to what extent do datasets generated by this pipeline enable reinforcement-learning fine-tuning of a compact VLM to improve grounding performance on desktop environments?**

1.4 Objectives

(1) Design an end-to-end desktop annotation pipeline that yields pixel-accurate bounding boxes and concise instructions with high human agreement. (2) Fine-tune a compact VLM with a lightweight RL scheme to convert near-misses into hits. (3) Evaluate whether the pipeline and RL fine-tuning improve desktop grounding performance (see Chapter 3 for operationalisation and metrics).

1.5 Contributions

- **End-to-end annotation pipeline.** A production-ready system that detects, describes, visualises, validates, and exports UI element annotations at scale. It operates without per-image manual intervention and supports adjustable detail levels for different use cases. The annotations created by this pipeline have been verified manually with an accuracy of 98.77% over 1,740 elements.
- **AccuAnnotate-2B (ours).** A 2B-parameter VLM for desktop GUI grounding that combines crop-interleaved prompting with RL fine-tuning (distance shaping, EMA baseline, entropy, warm-up/cooldown, τ-schedule). On the ScreenSpot desktop subset with greedy decoding, AccuAnnotate-2B improves cumulative hit-rate AUC and reduces distance-to-box relative to ShowUI-2B (e.g., +16.95% AUC and -9.08% DTB; see Section 4), converting near-misses into hits and lowering invalid-format rates.

Together, these components provide a practical bridge from *detect* $\rightarrow$ *describe* $\rightarrow$ *validate* $\rightarrow$ *export* $\rightarrow$ *fine-tune* demonstrating the ability of the data collection pipeline to generate correct training labels, and show that the VLM we train achieves better results in desktop environments.

CHAPTER 2

Literature Review

2.1 General VLMs

Visual Language Models (VLMs) refer to AI models that process both visual and text inputs, often by combining a vision encoder with a language model. The first landmark work, CLIP [21], demonstrated that prior state-of-the-art computer vision systems were trained to predict a fixed set of object categories, thereby restricting their generality and usability. They trained CLIP on a dataset of 400 million image–text pairs, demonstrating the potential of contrastive learning to align visual and textual representations. It was instrumental in popularising the concept of using contrastive learning to create shared embeddings for images and text. However, despite its pivotal role, CLIP revealed research gaps, notably the need for even larger and more diverse datasets.

A year after CLIP's release, in early 2022, Google published work aimed at addressing these gaps identified in the CLIP paper. They recognised that existing VLMs lacked the scale needed for real-world tasks and therefore released the 540 billion-parameter Pathways Language Model (PaLM) [22]. This model achieved top-notch few-shot learning results in various language benchmarks. In 2023, they further improved the model to include robotics tasks, making it a 562 billion-parameter model named PaLM-E [23].

OpenAI released GPT-4V (ision) in 2023. It is a large multimodal model with enhanced vision capabilities. GPT-4V supports various file formats and achieves promising results across multiple benchmarks. Although OpenAI did not disclose GPT-4V's technical details, the model has been independently evaluated by several researchers. For example, Yang et al. [24] assessed the capability of GPT-4V in different fields using a preliminary method. Many published papers cite this specific VLM model, showing its popularity in this field.

Interestingly, the authors of the RegionGPT paper identified that, whilst integrating VLMs with LLMs drove rapid progress, these models still struggled with detailed regional visual understanding. To address this gap, they developed RegionGPT [25], which enhances region-level captions. They also identified the **limitations**. They mentioned that (1) their task-guided instruction prompt sometimes did not restrict the response format well, and (2) the evaluation of object classification could be reformulated as the semantic similarity between the prediction and ground truth name via a pretrained text encoder.

These models have demonstrated the potential of VLMs and their wide range of applications.

Transformer backbones for visual encoders. Many modern VLMs rely on the Vision Transformer (ViT), which showed that a pure transformer over image patches can match or exceed convolutional networks when pre-trained at scale. ViT's patch embeddings and global self-attention have since become standard for contrastive and multimodal training [26].

Multimodal LLMs for general perception. Building on CLIP-style encoders, multimodal large language models (e.g. GPT-4V [18]) provide a generic interface for screenshot/image understanding and instruction following, but still require explicit grounding mechanisms to map language to precise on-screen targets; this motivates the UI-specific techniques reviewed below.

2.2 Usage of VLMs in UI

The application of VLMs in user interface (UI) fields has advanced significantly in the last few years. The use of VLMs in UI-related fields can be summarised into several categories.

2.2.1 UI Understanding and Grounding Abilities

Understanding pixel-based UI is a crucial task for various interaction applications, including accessibility and UI automation. Grounding ability means the ability to link words to pixels.

Prior UI modelling often relied solely on a screen's structural information, as highlighted in a Google Research paper [27]. They then built Spotlight – a novel VLM architecture capable of performing mobile UI tasks, such as few-shot learning, fine-tuning, and multi-task learning. Spotlight introduced a "focus region" mechanism, which takes as input a tuple of (screenshot, region of interest, task description). Spotlight achieved top-tier results on multiple benchmarks.

A subsequent study by Google DeepMind identified that infographics, more specifically charts, diagrams, and the like, which distil complex data into simple visuals, were challenging to incorporate into a single UI model. They built ScreenAI, a VLM for understanding complex UI and infographics [28]. This model achieved new state-of-the-art results on several UI benchmarks. Both Spotlight and ScreenAI focused exclusively on pixel-only data, revealing a gap in their methods; for example, a multimodal approach that also uses XML structural files might perform better.

From this perspective, introducing multimodal approaches may enhance the accuracy of the models. And for that, a research predating the GenAI and VLM boom highlighted that leveraging multimodal UI features was complex due to a lack of high-quality labelled data. Hence, they introduced UIBert [10], a transformer-based joint image–text model pre-trained on large-scale unlabelled UI data to learn generic UI representations. It outperformed strong baselines on nine UI tasks by up to 9.3% accuracy, demonstrating improved grounding of UI elements to language. This method can serve as a reference for improving the grounding capabilities of current VLMs.

Another recent study on improving grounding abilities identified that prior UI grounding models relied on developer-provided UI metadata to detect on-screen elements, but found that such metadata was often unavailable or incomplete. They developed LVG [29], a Layout-guided Visual Grounding model, which leveraged the structured nature of UIs to improve grounding accuracy. LVG exploited this by incorporating layout-guided contrastive learning. During training, the model learned to distinguish target UI components from distractors by using both visual features and their spatial organisation on the screen. The LVG faced several **limitations**, as the authors suggested in the paper. The model was developed and evaluated only on Android app screens so that it may struggle with the denser, more varied layouts of desktop or web UIs. Additionally, the model was trained and tested only on elements and expressions that were consistently present on the pages. In real-world scenarios, users may refer to UI elements that are not currently visible on the screen.

2.2.2 Conversational VLMs for UIs

Another direction of UI & VLM research is interactive UI agents, which could answer free-form questions about UIs.

A recent paper presented at the CORE A-rated IUI conference (2025) identified that many multimodal VLMs perform poorly on UI tasks due to a lack of specialised training data. They therefore introduced a

method to generate paired text–image data for the UI domain and fine-tune LLaVa [30] on their generated dataset [31].

Another approach, Ferret-UI [32], introduced by Apple's researchers in ECCV 2024, was a multimodal model that excelled at referencing specific UI components and reasoning about UI functionality. It was built on the Ferret model [33] and enhanced its understanding of mobile UI. Later, they released Ferret-UI 2 [34], accepted at ICLR 2025 (a CORE A* conference), which was optimised for all platforms and significantly outperformed its predecessor, Ferret-UI, across multiple benchmarks.

MobileVLM [35], on the contrary, noted that GPT-4V [18] and Gemini [36] provided very limited technical details, as they were closed-source models. MobileVLM was a competent multimodal VLM designed to run on mobile devices, achieving great performance compared to language models with 1.4 billion and 2.7 billion parameters [35]. The **limitations** of this work included training data from only 49 popular apps, which may not represent the full range of everyday Android use cases. More apps could be included in future work. Additionally, since some apps offer extra paid features, such as VIP content, the model may not fully capture every function. Finally, because app updates could randomly change page layouts and actions, the data might have some time-based **limitations**.

2.3 Vision–Language–Action (VLA) Models for Computer Use

Recent research moves beyond passive perception to acting on screens. There are two notable works that we want to introduce.

UI-TARS/2 (industrial, RL-centric). The UI-TARS line frames computer use as a native GUI-centred agent problem trained with large-scale rollouts and multi-turn reinforcement learning. The UI-TARS-2 technical report details a data flywheel for scalable trajectory generation, stabilised multi-turn RL, a hybrid GUI environment (file systems and terminals), and a unified sandbox for large-scale training [37]. Empirical results report consistent gains over earlier variants on long-horizon, information-seeking and software-use benchmarks, highlighting the efficacy of RL for planning and tool use in complex GUIs [37].

ShowUI (academic, data-efficient). In contrast, ShowUI treats VLA as a supervised, data-efficient problem with a compact model and a carefully curated GUI instruction dataset. It introduces UI-guided visual token selection (to prune redundant visual tokens) and interleaved vision–language–action streaming for multi-turn training/inference, reporting 75.1% zero-shot grounding on screenshot tasks

with a 2B model trained on 256k examples [19]. This line indicates that curation quality and UI-aware architectural bias can rival those of much larger systems.

These strands motivate accurate and scalable annotation: irrespective of the RL scale or model size, agents benefit from training data where targets are pixel-accurate and instructions are concise and unambiguous.

2.4 Screen Parsing and Grounding Pipelines

A recurring difficulty is mapping language instructions to exact on-screen targets. In practice, the community has progressed through three stages: (i) instrumented pipelines that rely on DOM/accessibility trees; (ii) hybrid computer-vision toolkits that detect elements from pixels but keep a human in the loop; and (iii) vision-only parsing that produces DOM-like structures directly from screenshots for downstream grounding. While this trajectory has steadily reduced dependence on platform metadata, open issues persist around cross-platform generalisability, the absence of standard interfaces between perception and language modules, and the **limited** availability of large, pixel-accurate desktop corpora.

From instrumented overlays to mark-based prompting. Early web agents exploited the DOM to obtain ground-truth element coordinates. Then they rendered overlays on the screenshot, allowing the model to choose a region ID rather than predict free-form coordinates. Set-of-Marks (SoM) formalised this idea by placing labelled, speakable marks (e.g. numbers/letters/masks/boxes) onto the image so a multimodal model such as GPT-4V could answer with a mark ID, which substantially improved localisation compared to unconstrained text outputs [17]. However, because the strongest variants rely on reliable access to element regions via browser or mobile instrumentation, the technique does not readily apply to uninstrumented desktop applications or legacy software, and its interface is primarily geared towards single-point selection rather than richer actions.

Hybrid pixel toolkits with human-in-the-loop. To reduce dependence on DOMs, GUI element detection toolkits emerged that operate directly on screenshots. UIED combined classical CV and deep models to segment widgets (buttons, text fields, icons), expose an interactive dashboard for correction, and export element metadata (position, size, class) for reuse [38]. These systems accelerated dataset creation and enabled controlled experiments on detection quality and class taxonomies. However, because they rely on manual cleanup and schema-specific categories, throughput remains bounded by review effort, transfer to unseen applications is limited, and exported annotations typically require additional processing before becoming training-ready, instruction-grounded labels.

Vision-only parsing for portable grounding. Recent pipelines aim to recover a DOM-like structure purely from pixels so that *any* screenshot (web, mobile, desktop) can be turned into a list of candidate, interactable regions with short textual descriptors. OmniParser exemplifies this approach: it curates web screenshots to train a dedicated icon/element detector, augments this with OCR and a captioning model to infer element semantics, and then supplies the resulting structured list to a multimodal LLM for decision-making [1]. Reported evaluations show that pairing GPT-4V with such proposals improves action grounding without platform metadata [17]; nevertheless, performance is sensitive to detector recall/precision and OCR quality in dense, high-resolution interfaces, web-trained detectors may not transfer cleanly to desktop themes and applications, and multi-stage pipelines can accumulate errors unless uncertainty and affordances are represented explicitly.

2.5 Domain-specific VLMs for UI

Recent research has targeted specific domains or challenges in UI understanding using VLMs.

A paper published in October 2024 by Xiaomi AI Lab identified that mobile UI examples constituted only a small fraction of general training datasets; hence, they introduced Mobile3M and MobileVLM [35]. Mobile3M was a large Chinese mobile dataset comprising 3 million UI pages and real-world interactions. MobileVLM, despite sharing its name with Meituan Inc.'s version, was a fine-tuned Chinese mobile VLM. As a result, MobileVLM outperformed existing state-of-the-art VLMs by 14.34% on the ScreenQA dataset.

Another notable direction was the use of VLMs for the semantic retrieval of UIs. A very recent paper in CHI '25 suggested that it was a challenge to explore the vast space of UI references using existing AI-based UI search methods. Hence, they introduced a multimodal large language model (MLLM) called S&UI [39] to interpret semantics from mobile UI images. The researchers of this paper mentioned that their approach could perform significantly better than existing UI retrieval methods for UI searching.

In another paper, Yan et al. [16] identified gaps in existing smartphone GUI navigation methods, noting that traditional methods struggled to generalise and lost visual detail when converting screens to text, whilst large multimodal models like GPT-4V often failed to localise actions accurately. To bridge these gaps, they introduced MM-Navigator, a system that directly leveraged screen images and natural language instructions, employing a set-of-mark (SoM) prompting method to map UI elements to numeric tags for precise interaction. They used a self-summarisation module to condense historical context

efficiently. Although evaluations on iOS and Android datasets showed promising high-level understanding and reasonable localisation accuracy, the approach still faced **limitations** in zero-shot error handling, annotation parsing, computational cost, and the need for more dynamic evaluation environments.

2.6 UI Datasets and Benchmark

In computer vision, UI understanding datasets received less attention compared to general image datasets.

Dataset landscape. Prior UI datasets span different screen domains (mobile, web, and—less commonly—desktop) and tasks ranging from screen summarisation and widget captioning to element/action grounding. Many mobile-first corpora (e.g., S2W, Widget Captioning, RICOSCA, MoTIF, RefExp) emphasise description or reference resolution on phone UIs, while several newer web-oriented sets (SeeClick Web, MultiUI, UGround-Web, UI REC/REG, Ferret-UI) scale up element grounding with automated annotation and multi-resolution assets. In contrast, our ACCUANNOTATE targets GUI screenshot grounding across Web, Mobile, and Desktop, supplying high-precision labels via a hint-constrained, crop-interleaved pipeline (Chapter 3). Table 2.1 summarises representative datasets and highlights where ACCUANNOTATE fits: multi-platform coverage including desktop, automated yet conservative annotation, and task focus on pixel-level grounding from screenshots.

Dataset	UI Type	Multi Res.	Auto Anno.	#Anno.	Task
Screen 2 Words [8]	Mobile	✗	✗	112k	Screen Summarization
Widget Captioning [9]	Mobile	✗	✗	163k	Element Captioning
RICOSCA [12]	Mobile	✗	✗	295k	Action Grounding
MoTIF [40]	Mobile	✗	✗	6k	Mobile Navigation
RefExp [10]	Mobile	✗	✗	20.8k	Element Grounding
SeeClick Web [2]	Web	✗	✓	271k	Element Grounding
MultiUI [41]	Web, Mobile	✓	✓	3M	Act. & Elem. Ground
UGround-Web [42]	Web	✓	✓	1.3M	Element Grounding
UI REC/REG [43]	Web	✓	✓	400k	Box2DOM, DOM2Box
Ferret-UI [32]	Mobile	✓	✓	250k	Elem. Ground & Ref.
AutoGUI [4]	Web, Mobile	✓	✓	704k	Functionality Ground & Ref.
AccuAnnotate (ours)	**Web, Mobile, Desktop**	✓	✓	**8k**	**Element Ground from GUI Screenshots**

TABLE 2.1: Comparison of UI datasets. "Multi Res.": multiple resolutions available. "Auto Anno.": automated annotation. Originally from AutoGUI [4]

.

AutoGUI [4], released earlier this year, identified that existing UI datasets either offered large-scale annotations with **limited** context or rich contextual descriptions on a small scale. Hence, they proposed

AutoGUI to fill the gap. AutoGUI was a pipeline that automatically annotated UI elements at scale. They used LLMs to infer functions of elements by comparing the UI change after simulated interactions with specific UI elements. They constructed a 704 k-parameter dataset, AutoGUI-704k, using their proposed method, which they reported greatly enhanced VLMs' UI grounding capabilities.

Prior work, such as widget captioning [9], identified a lack of natural language descriptions for UI elements; accordingly, they created a dataset of 61 285 elements from 21 750 unique UI screens. This research became the foundation for later studies on UI understanding. The authors acknowledged that whilst their model could generate semantically meaningful captions, it still faced several **limitations**. Their error analysis revealed that the model was often confused by nearby elements and similar visual appearances, resulting in miscaptions or overly generic descriptions. Additionally, the performance was constrained by the quality of the encoders for UI images and view hierarchies, indicating that further improvements in these components and an expanded, more varied dataset were necessary for better captioning accuracy.

The pix2code [44] paper highlighted the labour-intensive and error-prone nature of manually converting GUI designs into executable code, particularly across multiple platforms. It also highlighted the shortcomings of prior methods that relied on engineered heuristics for reverse-engineering UI layouts. To address these gaps, the authors introduced an end-to-end deep learning model that combined convolutional neural networks for visual feature extraction with recurrent neural networks for sequence generation, effectively learning to map GUI screenshots directly to a domain-specific language (DSL) representing the interface layout. Despite achieving promising results across iOS, Android, and web-based interfaces using synthesised datasets, the approach was **limited** by its reliance on synthetic data, a relatively modest model size, and the use of one-shot encoding, which restricted the ability to capture richer token relationships and scale to more complex languages.

ScreenSpot-Pro [3] introduced a large-scale high-resolution benchmark specifically tailored to professional desktop environments. Unlike prior UI datasets that concentrate on mobile interfaces or synthetic layouts, ScreenSpot-Pro comprises authentic screenshots from 23 widely used applications spanning development IDEs, creative suites, CAD/engineering tools, scientific analysis platforms, and office productivity software across Windows, macOS, and Linux systems. Each image is exhaustively annotated by domain experts to capture both iconographic and textual elements, covering hundreds of distinct icon classes and fine-grained text labels. Preliminary evaluations on this benchmark reveal a drastic performance drop for state-of-the-art VLM-based grounding models, with the best achieving

only 18.9% accuracy in combined icon and text localisation tasks. By establishing rigorous evaluation metrics and highlighting challenges such as small target sizes, dense layouts, and diverse resolution scales, ScreenSpot-Pro sets a new standard for advancing GUI perception models in real-world professional workflows.

Beyond mobile-centric resources, the community has developed interactive web/desktop suites to evaluate agents. *MiniWoB++* provides compact instruction-following tasks in synthetic web pages and remains a common starting point for pixel or DOM-based agents [45]. *Mind2Web* moves to real websites with 2,000+ open-ended tasks across 137 sites, exposing the variability and long-tail behaviours of the live web [14]. For desktop-class applications, *OSWorld* offers real computer tasks across operating systems and applications with execution-based evaluation, revealing sizeable **gaps** between human and model performance on open-ended computer use [15]. These suites are invaluable for end-to-end assessment, yet they do not by themselves constitute large corpora of static, annotated screenshots suitable for training perception modules.

Recent data pipelines aim to fill this gap. OmniParser reports semi-automatic curation of web screenshots with icon/element boxes distilled into a detector usable across platforms [1]. In parallel, data-centric curation in ShowUI demonstrates that careful resampling and balancing of GUI instruction data can yield strong grounding with comparatively small models [19]. Together with simulation-driven annotation at scale (e.g. AutoGUI [4]), and advancement in the new GPT-5 model [46] these works suggest a viable way to craft an automated annotation pipeline which is reliable and low-cost.

2.7 Conclusion/Discussion

Recent work confirms that Vision–Language Models have progressed from general image–text alignment (e.g. CLIP [21]) to increasingly specialised systems such as Spotlight [27], ScreenAI [28] and RegionGPT [25]. These models demonstrate strong screen-level understanding yet leave three persistent gaps: *(i)* insufficient coverage of desktop UIs, *(ii)* limited modelling of hierarchical relationships among interface elements, and *(iii)* their short region captions rarely say what each part does. Follow-up efforts like LVG [29] and Ferret-UI 2 [34] begin to address grounding and conversational control, but they still assume mobile layouts and rely on metadata that is often absent in professional software.

CHAPTER 3

Methodology

3.1 AccuAnnotate: A Data Collection Pipeline

3.1.1 System Overview

As Figure 3.1 suggests, ACCUANNOTATE is a three–tier application comprising a Flask backend, a Python annotator, and a user-friendly frontend.

Backend (Flask). The REST API (`app.py`) manages image annotation I/O and batched jobs with server-sent events (SSE). It uses SQLite (`data/metadata.db`) for data persistence.

Annotator. The annotator (`utils/annotator.py`) composites pre-processing with OmniParser, crop generation, strict prompts, functions to utilise state-of-the-art VLMs, and post-processing (snapping, clamping, and deduplication).

Frontend. A single HTML/JS/CSS interface (`templates/index.html` plus vanilla JS and CSS) supports upload, folder-structured browsing, filtering, preprocessing preview, LLM annotation, JSON paste/validation, visualisation overlays, batch annotate, move/delete, and export.

3.1.2 Storage and Layout

Filesystem. Images live under `data/images/`. Annotations are stored in `data/annotations/<subfolders>/<stem>.json`, preserving the image folder structure. Upon initial access, legacy flat files are automatically migrated.

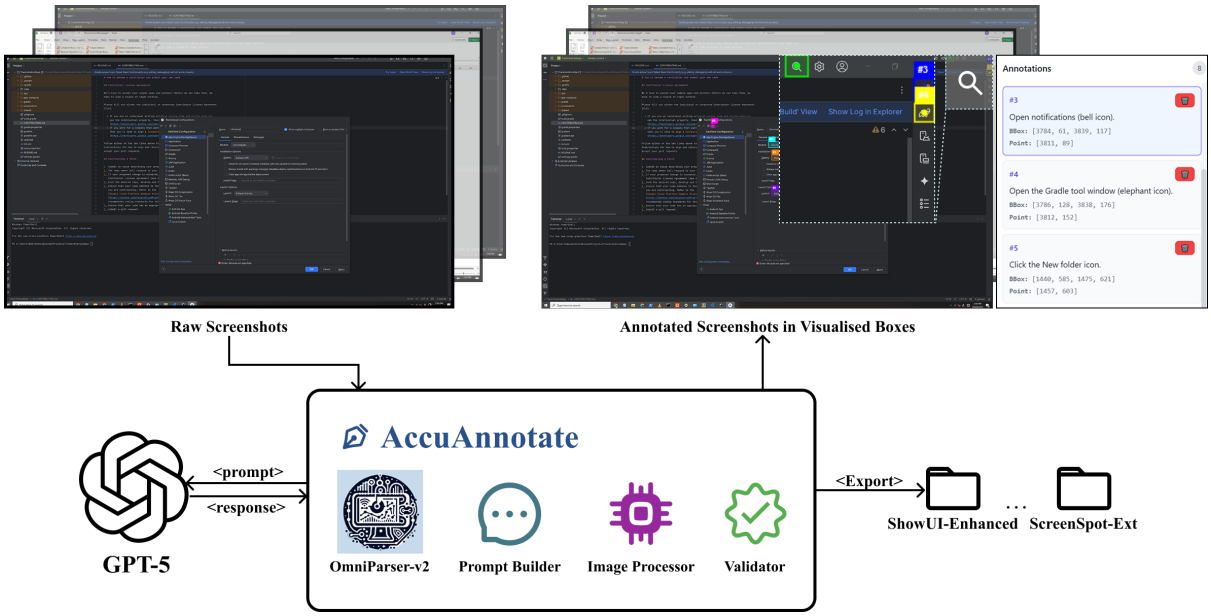

FIGURE 3.1: ACCUANNOTATE workflow, taking in raw screenshots as inputs, leverages components to process the screenshots and exports training/evaluation data in compatible dataset formats

Database. SQLite at `data/metadata.db` tracks folders (including empty chains), image sizes, and annotation status. On an empty DB, the system auto-indexes from the file system and subsequently relies on the DB.

3.1.3 Data Model

We standardise on absolute pixel geometry for best alignment (normalised values increase vagueness of model outputs). Annotation files follow:

```
{
  "img_size": [1920, 1080],
  "element": [
    {
      "instruction": "Open Settings",
      "bbox": [10, 20, 120, 60],
      "point": [65, 40],
      "source_id": 12
    }
  ]
```

```
}
```

All coordinates are *integers in absolute pixels* and exporters convert to normalised $[0, 1]$ when needed.

3.1.4 End-to-End Pipeline

For each image:

(1) **Select/Upload.** User uploads or picks an image from the folder-aware browser.

(2) **Preprocess (default-on).** OmniParser detects candidates and returns a normalised set of $[\text{bbox}, \text{point}, \text{confidence}]$. Candidates are ranked and pruned by confidence, IoU, and near-duplicate centres; stable `source_id`s are assigned and hints are sorted in reading order (top-to-bottom, left-to-right).

(3) **Crop Generation.** For selected hints, generate *tight* crops and (when row-like structure is detected) a second *directional/padded* crop to include nearby label text. A deterministic seed (comprising the filename, initial bytes, and hint geometry) ensures reproducibility. Crops are resized to a configurable long side, and each crop is preceded by a textual marker `<crop id=K type=...>`.

(4) **Prompting.** The prompt injects `<img_size>[W, H]</img_size>`; provides a `<hints>` array of candidate boxes/points (with optional shard indices); explains crop markers; instructs the model to choose *only* among hints, keep exact bbox equality, enforce bounds, uniqueness, and sorting; and return JSON only.

(5) **OpenAI Call.** The annotator uses the OpenAI Python SDK (chat or responses API). An optional code interpreter tool can be enabled to further increase grounding accuracy. If truncation is detected, a single retry is issued with a higher token budget; errors return explicit messages.

(6) **Post-process.** Parsed JSON is validated and *snapped* to exact hint boxes via `source_id` or highest-IoU match; centres are clamped inside boxes; duplicates are pruned; outputs are re-sorted for determinism and saved.

(7) **Visualise.** Server-side overlays (PIL) and frontend canvas overlays provide QA.

3.1.5 Preprocessing with OmniParser

Local mode. `omniparser_local.py` runs YOLOv8 detection with optional Florence-2 captioning (GPU recommended).

Remote mode. If `OMNIPARSER_URL` is set, the annotator calls an authenticated HTTP endpoint.

Ranking/limiting. Candidates are confidence-sorted, pruned by IoU and near-duplicate centres, bounded by a top-k from env, assigned stable IDs, and ordered by centre for reading order. A row-like heuristic triggers directional crops.

3.1.6 Prompting Constraints

The model is instructed to:

(1) select elements strictly from `<hints>` (no free-form coordinates),
(2) keep bboxes *exactly* equal to the chosen hint,
(3) enforce image bounds and integer coordinates,
(4) avoid duplicates and sort by reading order, and
(5) emits valid JSON only.

Interleaving `<crop ...>` markers with images binds textual instructions to pixels, thereby improving grounding.

3.1.7 Inference and Error Handling

Both chat and responses APIs are supported; timeouts and truncation trigger a single controlled retry with a higher token budget. Parsing surfaces explicit errors. Snapping prefers `source_id`; otherwise, max-IoU is used. Centres are clamped to $[\mathrm{x1}, \mathrm{x2}] \times [\mathrm{y1}, \mathrm{y2}]$, duplicates removed, and results persisted.

3.1.8 Backend API Surface

`GET /api/images`	Paginated DB-backed listing
`GET /api/folders`	Folder tree
`POST /api/upload`	Upload images (size-limited, `secure_filename`)
`GET /api/image/<path>`	Stream image with cache headers
`POST /api/preprocess/<path>`	Return candidate boxes/points (no LLM)
`POST /api/annotate/<path>`	Run full pipeline and persist JSON
`GET /api/annotation/<path>`	Read annotation
`PUT /api/annotation/<path>`	Update annotation
`DELETE /api/annotation/<path>/element/<i>`	Delete element i
`POST /api/batch-annotate`	Launch async job; `GET /status/<job_id>` and SSE `/stream/<job_id>` for progress
`POST /api/export`	Export (ShowUI-Desktop); `GET /api/download-export` for ZIP
`POST /api/deduplicate`	Remove duplicates by base filename

3.1.9 Frontend and Visualisation

Folder-aware browsing supports search, filters (annotated/not), sorting, selection mode, select-all, delete, move-to, and view toggles. Per-image tools include preprocessing preview, LLM annotation, overlay controls, and a "Paste JSON" modal with validation and explicit pixel-format examples. Bulk tools include force re-annotation, SSE progress bars, bulk deletion, and an export dialogue (with options for format/split/, and ZIP).

3.1.10 Export and Utilities

The ShowUI-Desktop exporter (`scripts/export_showui_desktop.py`) retrieves selected annotated items via the DB, copies images with folder structure, converts absolute pixels to normalised $[0, 1]$, and writes `metadata/hf_<split>.json` plus a Parquet file when `pandas`/`pyarrow` are available. Utilities include `import_data.py` (re-index), `migrate_annotations.py` and `fix_annotation_paths.py` (legacy path migration), `flatten_showui_dataset.py` (flattening), and `EXPORT_FORMATS.md` (adding new export formats).

3.1.11 Key environment knobs

OpenAI: `OPENAI_API_KEY`, `OPENAI_MODEL`, `OPENAI_MAX_COMPLETION_TOKENS`, `OPENAI_MAX_TOKENS`, `OPENAI_SERVICE_TIER`, `OPENAI_TIMEOUT_SECONDS`, `OPENAI_ENABLE_CODE_INTERPRETER`.

Preprocess/Annotator: `ANNOTATOR_PREPROCESS_ENABLE`, `ANNOTATOR_PREPROCESS_MAX_ELEMENTS`, `ANNOTATOR_MAX_SHARDS`, `ANNOTATOR_SHARD_TOPK`, `ANNOTATOR_DUAL_CROP_TOPK`, `ANNOTATOR_CROP_PAD_PX`, `ANNOTATOR_TEXT_PAD_PX`, `ANNOTATOR_CROP_LONG_SIDE`, `ANNOTATOR_SHARD_SEED`.

OmniParser: `OMNIPARSER_URL`, `OMNIPARSER_API_KEY`, `OMNIPARSER_TIMEOUT`, `OMNIPARSER_MIN_CONF`, `OMNIPARSER_CONF_THRESHOLD`.

3.1.12 Design Rationale

Three decisions drive label fidelity:

(1) **Absolute pixels instead of normalised** We maintain tight geometric agreement during curation, converting to normalised coordinates only at export.

(2) **Hint-constrained labelling with crop interleaving.** The model is restricted to OmniParser hints and guided by interleaved crop markers; outputs are snapped to hints to mitigate hallucinations and enforce geometric consistency.

3.1.13 Pseudocode Summary

Algorithm 1: Per-image annotation pipeline

```
Input: img_path
Output: elements = [{instruction, bbox, point, source_id}]

img    = load_image(img_path)
hints  = OmniParser(img)  // [{bbox, point, conf}]
hints  = RankAndPrune(hints, by=conf, IoU, near-duplicate centers)
crops  = GenerateCrops(img, hints, deterministic_seed=True)
prompt = BuildPrompt(img_size, hints, crops, strict_constraints=True)
resp   = OpenAI(prompt, retry_if_truncated=True)
raw    = ParseJSON(resp)
```

```
8: elements = []
9: for e in raw:
10:   h = MatchHint(e, hints, by=source_id or max-IoU)
11:   c = ClampCenterInside(h.bbox, e.point)
12:   elements.append({instruction=e.instruction, bbox=h.bbox,
                       point=c, source_id=h.id})
13: elements = Deduplicate(elements)
14: SaveAnnotation(img_path, elements, img_size, version="2.1+")
15: RenderOverlays(img_path, elements)
```

3.2 AccuAnnotate-2B VLM Model

To validate the effectiveness of our data collection method and to address the limitation that the original ShowUI-2B's performance is capped by its data size in desktop environments, we introduce the ACCUANNOTATE-2B model.

3.2.1 Problem Setup

We target pixel-level grounding for desktop graphical user interfaces (GUIs): given a screenshot I and a natural-language instruction q, the agent must predict a clickable point $\hat{\mathbf{p}} = [\hat{x}, \hat{y}] \in [0, 1]^2$ on the screen corresponding to the described UI element. Each training sample is a triplet.

$$s = (I, q, \mathbf{p}^\star), \qquad \mathbf{p}^\star = [x^\star, y^\star] \in [0, 1]^2,$$

where $\mathbf{p}^\star$ denotes the ground-truth normalised coordinate (or the centre of a ground-truth bounding box). The policy π_θ is a vision-language model (VLM) that autoregressively generates a short textual sequence which is parsed into $\hat{\mathbf{p}}$.

3.2.2 Base Model and Data

We fine-tune the *ShowUI-2B* backbone (Qwen2-VL style) using 400 unique desktop UI screenshots paired with 2,630 instruction–target tasks. To ensure robust supervision and easy integration of different data sources, we implemented a unified loader that supports: (i) **ShowUI-style JSON**: paths to images with elements containing an `instruction` and a normalised `point`; (ii) **Parquet-based screens**: columns

mapping image paths/bytes, instructions, and either bounding boxes or points; and (iii) **Free-form JSON**: a light schema with `image_context` and `target_bbox` that we normalise to $[0,1]^2$ using image dimensions. For samples annotated by bounding boxes $[x, y, w, h]$ or $[x_1, y_1, x_2, y_2]$, we convert to the normalised center $\mathbf{p}^\star$ and clamp to $[0,1]^2$.

3.2.3 Prompted Coordinate Policy

A strict, uniform chat template is used to condition the model:

- The screenshot I is embedded via the processor with a constrained token budget using `min_visual_tokens` and `max_visual_tokens` (converted internally to pixel budgets).
- The instruction q is augmented with a *schema constraint*: "*Return exactly two numbers in square brackets like [x, y] with both x and y in* $[0,1]$." This markedly reduces invalid generations in early training.

At each step, the model generates a short sequence $y_{1:T}$ that is parsed by a tolerant PARSECOORD function supporting lists, tuples, percentages, and (when image size is known) pixel coordinates, all mapped to $[0,1]^2$.

3.2.4 Reward Shaping

We design a simple yet effective reward function that balances point accuracy with format compliance. Given a predicted coordinate $\hat{\mathbf{p}} = [\hat{x}, \hat{y}]$ and the ground-truth center $\mathbf{p}^\star = [x^\star, y^\star]$, we compute the normalized Euclidean distance $d = \|\hat{\mathbf{p}} - \mathbf{p}^\star\|_2$ and define:

$$r(\hat{\mathbf{p}}, \mathbf{p}^\star) = \begin{cases} \mathbb{1}(d < \tau) - \alpha_{\text{dist}} \, \min(d, 0.5) & \text{if } \hat{\mathbf{p}} \text{ is valid,} \\ -1.0 & \text{otherwise (NaN or unparseable).} \end{cases} \tag{3.1}$$

The reward consists of two components:

- **Binary success signal**: $\mathbb{1}(d < \tau)$ awards $+1.0$ if the prediction falls within a threshold radius τ of the target (default $\tau = 0.06$), encouraging precise localization.

- **Linear distance penalty**: $-\alpha_{\text{dist}} \min(d, 0.5)$ provides a shaped gradient even for unsuccessful attempts, guiding the policy toward the target. We cap the penalty at distance 0.5 to avoid over-penalising distant predictions. We use $\alpha_{\text{dist}} = 1.0$ by default.

The final reward is normalised to $[-1.0, 1.0]$ for numerical stability. Invalid predictions (model generates text that cannot be parsed into two coordinates) receive a score of $r = -1.0$, which discourages format errors without requiring an explicit format-compliance term.

3.2.5 Policy Gradient Objective

Following the REINFORCE paradigm [47, 48], we maximise the expected reward by minimising the negated policy gradient loss over generated token sequences. Let $y_{1:T}$ denote the tokens generated for sample $s = (I, q, \mathbf{p}^\star)$. Define the per-sequence negative log-likelihood over all generated tokens (excluding padding):

$$\ell_{\pi_\theta}(y_{1:T} \mid s) = \frac{\sum_{t=1}^{T} m_t \, \text{NLL}(y_t \mid y_{<t}, I, q; \theta)}{\sum_{t=1}^{T} m_t}, \tag{3.2}$$

where $m_t = \mathbb{1}(y_t \neq \texttt{pad})$ is a binary mask that excludes padding tokens. The model is trained to generate short sequences, such as "`[0.42, 0.73]`," with minimal verbosity. In practice, most or all non-padding tokens encode the coordinate prediction.

EMA-Normalised Advantage. To stabilise learning with small batch sizes (`batch_size=1`), we normalise advantages using an *exponential moving average (EMA)* of reward statistics accumulated across training steps. Let $\bar{r}_{\text{EMA}}$ and σ_{EMA} denote the EMA baseline and standard deviation, updated at each step t as:

$$\bar{r}_{\text{EMA}}^{(t)} = \beta \, \bar{r}_{\text{EMA}}^{(t-1)} + (1 - \beta) \, r^{(t)}, \tag{3.3}$$

$$(\sigma_{\text{EMA}}^{(t)})^2 = \beta \, (\sigma_{\text{EMA}}^{(t-1)})^2 + (1 - \beta) \, (r^{(t)} - \bar{r}_{\text{EMA}}^{(t)})^2, \tag{3.4}$$

where $\beta = 0.9$ is the EMA decay factor. The advantage for sample i is then:

$$A_i = \frac{r_i - \bar{r}_{\text{EMA}}}{\sigma_{\text{EMA}} + \epsilon}, \tag{3.5}$$

with $\epsilon = 10^{-8}$ for numerical safety. If $\sigma_{\text{EMA}} \leq 10^{-6}$ (rewards become constant), we centre but do not normalise. For additional robustness, we clip advantages to $[-C_{\text{adv}}, C_{\text{adv}}]$ with $C_{\text{adv}} = 3.0$.

The *policy gradient loss* is then

$$\mathcal{L}_{\text{pg}} = \frac{1}{B} \sum_{i=1}^{B} A_i \, \ell_{\pi_\theta}(y_{1:T}^{(i)} \mid s_i). \tag{3.6}$$

Entropy Bonus. To encourage exploration and prevent premature collapse to a deterministic policy, we add an entropy bonus computed over all generated (non-padding) tokens:

$$H_\theta = \frac{1}{B} \sum_{i=1}^{B} \frac{\sum_{t=1}^{T} m_t^{(i)} \, H\big(\pi_\theta(\cdot \mid y_{<t}^{(i)}, s_i)\big)}{\sum_{t=1}^{T} m_t^{(i)}}, \tag{3.7}$$

where $H(\cdot)$ is the Shannon entropy of the token distribution. The entropy coefficient λ_H is scheduled from an initial value (e.g., 0.01) to a final value (e.g., 0.0) over training to transition from exploration to exploitation.

Optional KL Regularisation. Optionally, to prevent the policy from drifting too far from a pretrained or earlier checkpoint, we include a KL penalty to a frozen reference policy π_{ref}:

$$\text{KL}_\theta = \frac{1}{B} \sum_{i=1}^{B} \frac{\sum_{t=1}^{T} m_t^{(i)} \, \text{KL}\left(\pi_\theta(\cdot \mid y_{<t}^{(i)}, s_i) \,\|\, \pi_{\text{ref}}(\cdot \mid y_{<t}^{(i)}, s_i)\right)}{\sum_{t=1}^{T} m_t^{(i)}}, \tag{3.8}$$

When the KL divergence is computed in closed form from the token-level logits, the reference policy is turned on by default ($\lambda_{\text{KL}} = 0$), which keeps the model from drifting too far and stabilises training. Still, it can be activated with an 8-bit quantised reference model for additional stability.

The full training objective is

$$\mathcal{L} = \mathcal{L}_{\text{pg}} - \lambda_H \, H_\theta + \lambda_{\text{KL}} \, \text{KL}_\theta. \tag{3.9}$$

We use $\lambda_H = 0.01$ (annealed to 0.0) and $\lambda_{\text{KL}} = 0.0$ (optionally up to 10^{-4}) by default.

Adaptive KL Control. When KL regularisation is enabled, we adapt λ_{KL} online using a simple multiplicative controller to maintain a target KL divergence $\text{KL}_{\text{target}} = 0.1$. Every $N_{\text{adapt}} = 50$ steps, we adjust:

$$\lambda_{\text{KL}}^{(t+1)} = \begin{cases} \lambda_{\text{KL}}^{(t)} \times \rho & \text{if } \text{KL}_\theta^{(t)} > 1.3 \, \text{KL}_{\text{target}}, \\ \lambda_{\text{KL}}^{(t)} / \rho & \text{if } \text{KL}_\theta^{(t)} < 0.77 \, \text{KL}_{\text{target}}, \\ \lambda_{\text{KL}}^{(t)} & \text{otherwise,} \end{cases} \tag{3.10}$$

where $\rho = 1.5$ is the adaptation rate, and we clamp $\lambda_{\mathrm{KL}} \in [0.0, 0.5]$. This maintains a soft constraint on policy drift while allowing exploration when reward signals are strong.

3.2.6 Decoding, Safety, and Robustness

Generation uses either greedy decoding or sampling (top-k/top-p, temperature T), with a warm-up phase that forces greedy decoding for the first N steps to reduce invalid formats. We detect rare "meltdown" states (excessive entropy or KL) and trigger a temporary safety cooldown: force greedy decoding for a few steps and decay the temperature toward a floor $T_{\min}$.

Format-constrained decoding. During training we enable a lightweight regex/FSM-gated decoder that only admits strings matching `\[\s*<num>,\s*<num>\s*\]` (with optional whitespace and signs). On mismatch we trigger a local fallback to greedy emission of numeric tokens. This is the "FSM decoder" referenced in Section 4.8.

To improve numerical robustness, we (i) cast pixel tensors to the model's parameter dtype, (ii) sanitise logits with `nan_to_num`, and (iii) clip global gradients at a fixed norm.

3.2.7 Training Procedure

We train on a single GPU in bf16 (or 8-bit quantised weights when requested), using AdamW with a small learning rate and gradient clipping. Each epoch consists of a fixed number of optimisation steps; at every step, we:

(1) **Batch construction**: randomly sample an element $(I, q, \mathbf{p}^\star)$ and build a strict prompt with the embedded image and schema constraint.
(2) **Rollout**: decode $y_{1:T}$, parse $\hat{\mathbf{p}}$, and compute r via (3.1).
(3) **Loss**: compute seqNLL on generated tokens, form A with EMA baseline/variance, and assemble $\mathcal{L}$ in Equation (3.9) with entropy and (optional) KL.
(4) **Update**: backpropagate (with optional grad accumulation), clip, and step the optimiser.

We log scalar summaries (loss, reward, entropy, KL divergence, and invalid-parse rate) to TensorBoard and also stream per-step JSONL records for auditability. And checkpoints are saved periodically.

3.2.8 Evaluation Within Training (for Model Selection Only)

For early feedback and model selection, we run a lightweight *ScreenSpot* subset evaluation at a fixed cadence. By default, we filter to the `desktop` environment and cap the evaluation set size (e.g., 200 samples) for speed. A prediction counts as success only if the parsed $\hat{\mathbf{p}}$ falls inside the ground-truth bounding box. Decoding is greedy for determinism. This signal is *not* optimised directly; it is used only to monitor progress and select checkpoints.

3.2.9 Schedules and Hyper-parameters

We train a small-batch policy-gradient objective with AdamW (learning rate 5×10^{-6}), optional gradient accumulation, and global gradient clipping (max norm $= 1.0$). Unless otherwise noted, each run uses E=2 epochs with S=1000 optimisation steps per epoch (T_{train}=$E \times S$), matching the ablation budget in Section 4.8. Training uses sampling; evaluation is greedy for determinism (see Section 3.2.6).

Linear schedules. We apply *linear* schedules over the total number of training steps T_{train} to the temperature and the entropy coefficient:

$$\text{Temp}(t) = T_{\text{start}} + \big(T_{\text{end}} - T_{\text{start}}\big)\,\frac{t}{T_{\text{train}}}, \tag{3.11}$$

$$\lambda_H(t) = \lambda_H^{\text{start}} + \big(\lambda_H^{\text{end}} - \lambda_H^{\text{start}}\big)\,\frac{t}{T_{\text{train}}}. \tag{3.12}$$

Success radius. By default we use a fixed success radius $\tau = 0.06$ during training and evaluation. Only in the ablation study (Section 4.8, `adaptive_tau`) do we schedule

$$\tau(t) = \tau_{\text{start}} + \big(\tau_{\text{end}} - \tau_{\text{start}}\big)\,\frac{t}{T_{\text{train}}}$$

to test sensitivity to the radius.

KL regularisation. KL is *enabled by default* with λ_{KL}=0.02 to a frozen 8-bit quantised reference policy; the `no_kl` ablation disables it. When desired, we adapt λ_{KL} online using the multiplicative controller described under *Adaptive KL Control*; otherwise it remains fixed.

Other defaults. Unless otherwise stated: batch size $= 1$ (with optional gradient accumulation), mixed precision (bf16 where available), and the regex/FSM-gated decoder is used only during training to stabilise formatting (evaluation remains greedy; see Section 3.2.6).

3.2.10 Implementation Details and Reproducibility

We seed all random number generators; enable cuDNN/TF32 fast paths when available; and support auto-resume from the latest `rl_ckpt_epoch*` directory. Each checkpoint includes model weights and configuration; the training state (optimiser and global step) is also persisted when requested. Prompts, decoded samples, and reward diagnostics (mean/min/max, invalid parse rate) are logged for qualitative inspection. The entire pipeline is single-GPU, self-contained, and requires only a dataset directory with images plus a metadata split file.

Summary. In short, we fine-tune a compact VLM for GUI grounding via a carefully engineered policy-gradient objective with (i) distance-based reward shaping, (ii) EMA-normalised advantages, (iii) entropy regularisation and optional reference KL, (iv) decoding warm-up and safety cooldown, and (v) progressive schedules for temperature, entropy weight, and success radius. The combination creates a stable and sample-efficient reinforcement learning process using a modest collection of 400 desktop screenshots and 2,630 tasks, while adhering to a strict output format that is essential for reliable GUI actions.

CHAPTER 4

Experiments

4.1 Goals

We evaluate whether the proposed pipeline in Section 3.1 and RL fine-tuning improve pixel-level grounding on desktop GUIs.

Success criteria. We evaluate on desktop-focused ScreenSpot splits (and ScreenSpot-Pro when specified) using: (i) click-inside-box Success ↑, (ii) point L2 ↓, (iii) Invalid-format rate ↓, (iv) AUC (cumulative hit rate) ↑, and (v) DTB (px) ↓. Unless noted, evaluation is strictly held out and uses greedy decoding for determinism. Uncertainty is reported as ±95% bootstrap CIs over items, and paired improvements are tested via bootstrap on per-item deltas.

4.2 Dataset Construction & Pipeline Settings

We construct the training data with the ACCUANNOTATE pipeline described in Section 3.1, then export to a ShowUI-Desktop–compatible format for RL fine-tuning.

Sources and counts. Our RL training set contains **400** desktop screenshots with **2,630** instruction–target tasks (points or boxes converted to centres). Internal model-selection evaluates on a capped ScreenSpot-desktop subset (see Section 3.2.6); all final results use the held-out benchmarks in Section 4.8 and Section 4.9.

Leakage control and deduplication. To prevent overlap between training exports and evaluation suites, we (i) deduplicate images by SHA-256 of raw bytes and by filename stems, (ii) exclude any item whose path/stem matches the ScreenSpot splits used for validation/testing, and (iii) remove near-duplicates

within training via perceptual hash and base-filename rules. This guarantees that no evaluation screenshot appears in training.

Pipeline configuration (frozen for training exports). Unless noted, we use the same annotator settings for all exports:

- **Preprocess:** `ANNOTATOR_PREPROCESS_ENABLE=true`, OmniParser v2 (remote if `OMNIPARSER_URL` set).
- **Candidate control:** confidence-sorted; IoU and near-centre pruning; top-k hints retained.
- **Crops:** tight crop per hint; row-like heuristic triggers a second directional/padded crop; deterministic seed binds crops to hint geometry.
- **Prompt:** `<img_size>[W,H]</img_size>` and a `<hints>` list; interleaved `<crop ...>` markers; schema instruction "return [x, y]".
- **Post-process:** snap to hint by `source_id` or max-IoU; clamp centre inside bbox; deduplicate; sort for determinism.

A concrete export profile used in our runs:

```
# AccuAnnotate export profile (training)
ANNOTATOR_PREPROCESS_ENABLE: true
ANNOTATOR_PREPROCESS_MAX_ELEMENTS: 50
ANNOTATOR_SHARD_TOPK: 25
ANNOTATOR_DUAL_CROP_TOPK: 10
ANNOTATOR_CROP_PAD_PX: 8
ANNOTATOR_TEXT_PAD_PX: 6
ANNOTATOR_CROP_LONG_SIDE: 512
ANNOTATOR_SHARD_SEED: "filename+geom"
OMNIPARSER_CONF_THRESHOLD: 0.30
OPENAI_MAX_COMPLETION_TOKENS: 128
EXPORT_FORMAT: "showui-desktop"
```

Quality assurance. Format validity is enforced at parse time; outputs are snapped to detector hints to preserve geometry. Agreement against a human annotator on $N{=}1{,}740$ tasks is reported in Table 4.1.[1]

[1] Table label updated to "Click-inside-box (same bbox)" to match the benchmark criterion.

Export artefacts and units. Exports write absolute-pixel annotations and convert to $[0, 1]^2$ only at save time when required by downstream tools. All distance-like metrics in Section 4.4 are computed in *pixels* by rescaling predictions/targets to the original image size.

4.3 Benchmarks and Splits

We use *ScreenSpot* as our primary evaluation suite and report a desktop-focused subset by default (for fidelity and speed), consistent with the model selection signal in training in Section 3.2.6. Unless noted, evaluation is strictly held-out: none of the evaluation screenshots appear in training exports.

Main setting. Desktop subset, N samples (capped for speed when specified). Greedy decoding for determinism. The success criterion follows the benchmark: a prediction is correct if it lies *inside* the ground-truth bounding box.

4.4 Metrics

We report discrete success, distance, and reliability metrics to comprehensively evaluate grounding performance.

Click-inside-box success (↑ higher is better). The *success rate* measures the proportion of predictions that fall strictly inside the ground-truth bounding box, matching the click-based interaction criterion used in agent evaluation. For sample i, let $(x_1^{(i)}, y_1^{(i)})$ and $(x_2^{(i)}, y_2^{(i)})$ denote the top-left and bottom-right corners of the ground-truth box, and let $(\hat{x}^{(i)}, \hat{y}^{(i)})$ denote the predicted click point:

$$\text{Succ} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\big[\, x_1^{(i)} \le \hat{x}^{(i)} \le x_2^{(i)} \ \wedge\ y_1^{(i)} \le \hat{y}^{(i)} \le y_2^{(i)} \,\big] \tag{4.1}$$

where $\mathbf{1}[\cdot]$ is the indicator function (1 if true, 0 otherwise).

Point L2 distance (↓ lower is better). The *L2 distance* (Euclidean distance) quantifies average pixel-level error between the predicted point and the ground-truth center. Let $\mathbf{p}^{\star(i)} = \big(\frac{x_1^{(i)}+x_2^{(i)}}{2}, \frac{y_1^{(i)}+y_2^{(i)}}{2}\big)$ denote the center of the ground-truth box and $\hat{\mathbf{p}}^{(i)} = (\hat{x}^{(i)}, \hat{y}^{(i)})$ the predicted point:

$$\text{L2} = \frac{1}{N}\sum_{i=1}^{N} \| \hat{\mathbf{p}}^{(i)} - \mathbf{p}^{\star(i)} \|_2 = \frac{1}{N}\sum_{i=1}^{N} \sqrt{(\hat{x}^{(i)} - x^{\star(i)})^2 + (\hat{y}^{(i)} - y^{\star(i)})^2} \tag{4.2}$$

L2 provides a continuous measure of accuracy that complements the binary success metric, penalizing near-misses proportionally to their distance from the target center.

AUC – Area Under the Curve (↑ higher is better). The *AUC* (Area Under the Hit-Rate Curve) measures the cumulative success rate across distance thresholds. For threshold $\tau \in [0, \tau_{\max}]$, we compute the hit rate as the proportion of predictions within τ pixels of the ground-truth center:

$$\mathrm{HitRate}(\tau) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\big[\, \|\hat{\mathbf{p}}^{(i)} - \mathbf{p}^{\star(i)}\|_2 \le \tau \,\big] \tag{4.3}$$

The AUC integrates this hit rate over all thresholds, normalized by the maximum threshold:

$$\mathrm{AUC} = \frac{1}{\tau_{\max}} \int_0^{\tau_{\max}} \mathrm{HitRate}(\tau)\, d\tau \tag{4.4}$$

In practice, we compute this via the trapezoidal rule over discrete thresholds. AUC summarizes accuracy across the entire precision spectrum—higher values indicate both better overall accuracy and more predictions clustered near the target.

DTB – Distance to Box (↓ lower is better). The *DTB* (Distance to [bounding] Box) measures the average minimum distance (in pixels) from each predicted point to the nearest point on or inside the ground-truth bounding box:

$$\mathrm{DTB} = \frac{1}{N} \sum_{i=1}^{N} d_{\mathrm{box}}\big(\hat{\mathbf{p}}^{(i)}, \mathcal{B}^{(i)}\big) \tag{4.5}$$

where $\mathcal{B}^{(i)}$ is the ground-truth box and

$$d_{\mathrm{box}}(\mathbf{p}, \mathcal{B}) = \begin{cases} 0 & \text{if } \mathbf{p} \in \mathcal{B} \\ \min_{\mathbf{q} \in \partial\mathcal{B}} \|\mathbf{p} - \mathbf{q}\|_2 & \text{otherwise} \end{cases} \tag{4.6}$$

Here $\partial\mathcal{B}$ denotes the boundary of box $\mathcal{B}$. DTB is zero for successful clicks and increases linearly with miss distance, providing a grounding-specific alternative to L2 that directly measures failure severity in the task's coordinate frame.

Invalid-format rate (↓ lower is better). The *invalid rate* quantifies the proportion of model outputs that fail to parse into valid (x, y) coordinates, indicating generation failures or formatting instability:

$$\mathrm{Invalid} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\big[\, \textsc{ParseCoord}(\text{output}^{(i)}) \text{ fails} \,\big] \tag{4.7}$$

Invalid predictions are excluded from distance metrics but counted as failures in success metrics. High invalid rates signal poor model calibration or inadequate output constraints.

Uncertainty quantification. We report $\pm 95\%$ confidence intervals (CIs) using the normal approximation over seeds. For a variant with k seeds, we compute the mean and standard deviation of the metric across seeds and construct the CI as $\bar{x} \pm 1.96 \cdot s/\sqrt{k}$, where s is the sample standard deviation. For paired model comparisons on per-item metrics, we compute the mean difference and assess whether it exceeds the combined standard error.

4.5 Annotation Quality and Label Granularity

4.5.1 Agreement with Human Annotations

We measured ACCUANNOTATE against a human annotator on $N{=}1{,}740$ tasks using the same click-inside-box criterion as our benchmark. Upon testing the accuracy of ACCUANNOTATE, we discovered that it achieves **98.77%** accuracy with an approximate 95% CI of ± 0.27 percentage points (normal approximation).

Set	Match Criterion	Accuracy (%)
ACCUANNOTATE vs. Human (1.7k)	Correct description of the image	**98.77** ± 0.27

TABLE 4.1: Annotation accuracy table (comparing to a human given the same bbox).

4.6 Baselines

- **ShowUI-2B (backbone).** Frozen base model evaluated with the same prompt and parsing.
- **AccuAnnotate-2B (ours).** RL fine-tuned model using the dataset generated by ACCUANNOTATE.

4.7 Hardware Setup

We ran all experiments on a single-GPU instance provisioned via Vast.ai. No multi-GPU or distributed training was used in the experiments.

Platform	Vast.ai
GPU	NVIDIA RTX Pro 6000 WS (96 GB VRAM)
CPU	48 cores
System RAM	255 GB

TABLE 4.2: Compute environment for all experiments.

4.8 Ablation Study

Setup. We evaluate on the desktop subset of *ScreenSpot* (N=334), using greedy decoding and the metrics in Section 4.4. We aggregate over multiple random seeds per variant (2–3 seeds as indicated) and report mean ± SD across seeds. Unless otherwise noted, Invalid-format is the proportion of outputs that fail to parse into coordinates.

Baseline. We treat *full* as the reference configuration. It includes sampling, the FSM format-constrained decoder, a fixed success radius τ, EMA baseline, entropy regularisation, KL regularisation, warm-up, and safety cooldown. The Δ columns in Table 4.3 are computed relative to this *full* setup.

Variant	Seeds	Succ (%)↑	L2↓	Invalid (%)↓	Train (min)↓	ΔSucc (pp)	ΔL2	Time×
`full`	2	71.11 ± 1.06	0.102 ± 0.008	0.00	28.0	0.00	0.000	1.00×
`no_fixed_tau`	3	68.56 ± 1.56	0.102 ± 0.004	0.00	60.1	-2.54	-0.000	2.15×
`no_distance`	2	68.71 ± 2.33	0.103 ± 0.009	0.00	28.0	-2.40	0.001	1.00×
`no_ema`	2	63.17 ± 5.08	0.112 ± 0.011	0.00	28.0	-7.93	0.010	1.00×
`no_entropy`	2	66.17 ± 1.27	0.107 ± 0.007	0.00	28.0	-4.94	0.004	1.00×
`no_kl`	2	69.01 ± 1.48	0.103 ± 0.006	0.00	25.9	-2.10	0.000	0.92×
`no_warmup`	2	67.51 ± 1.06	0.103 ± 0.003	0.00	28.0	-3.59	0.001	1.00×
`no_safety`	2	70.36 ± 3.39	0.104 ± 0.001	0.00	28.0	-0.75	0.002	1.00×
`greedy_only`	2	67.51 ± 1.48	0.100 ± 0.007	0.00	27.9	-3.59	-0.002	1.00×

TABLE 4.3: Ablations on the ScreenSpot desktop subset (N=334). Means ± SD across seeds. Δ columns are relative to `full`.

Findings.

- **Full is best on Success and format stability.** *Full* attains the highest seed-averaged Success $(71.11\% \pm 1.06)$ with competitive L2 (0.102 ± 0.008) and **0.00%** Invalid.
- **Dropping fixed τ is costly and slow.** `no_fixed_tau` lowers Success by $\mathbf{-2.54}$ pp (to $68.56\% \pm 1.56$) with no L2 benefit, while more than doubling wall-clock ($2.15\times$). This trade is not attractive under our budget.

- **EMA is critical.** `no_ema` is the most damaging toggle: Success $\mathbf{-7.93}$ pp (to $63.17\% \pm 5.08$) and the worst L2 (0.112 ± 0.011). The EMA baseline is thus a must-keep stabiliser.
- **Entropy regularisation helps exploration.** `no_entropy` hurts both Success (-4.94 pp) and L2 (+0.004), indicating that some entropy is beneficial for discovering good actions.
- **KL regularisation: modest accuracy drop, small speed gain.** `no_kl` yields -2.10 pp Success (to $69.01\% \pm 1.48$) with L2 near baseline (0.103 ± 0.006) and a slight speedup ($0.92\times$). *In this greedy-decode regime, KL is not required for format stability* (Invalid remains 0%), but it still provides a small accuracy edge.
- **Distance shaping matters.** `no_distance` drops Success by -2.40 pp and slightly worsens L2 (+0.001), supporting the value of the dense distance term.
- **Warm-up and safety cooldown are low-leverage here.** `no_warmup` reduces Success by -3.59 pp with negligible L2 change; `no_safety` is very close to Full (-0.75 pp, L2 +0.002).
- **Greedy-only objective optimises distance, not discrete hits.** `greedy_only` achieves the best L2 $(\mathbf{0.100} \pm 0.007)$ but sacrifices Success (-3.59 pp). If a downstream metric is distance-weighted rather than box-hit, this variant may be preferred.
- **Formatting stability.** All variants report **0.00%** Invalid under greedy decoding on this split, so decoder-level constraints mainly affect accuracy rather than parse validity in this setting.

Takeaways. Under a 2-epoch, 1000-steps/epoch budget on $N{=}334$ desktop items, the **must-keep** components are: fixed τ, EMA, entropy, and (for a small but consistent gain) KL. Removing fixed τ is strictly worse and $2.15\times$ slower; removing EMA is strongly harmful; turning off entropy reduces both Success and distance quality. KL can be disabled if $\sim$8% faster training is needed and a $\sim$2 pp Success drop is acceptable, noting that format stability remains intact under greedy decoding. If the target metric prioritises point precision, `greedy_only` offers the lowest L2 at a modest Success cost; otherwise, **Full** remains the most effective default.

4.9 Main Results (ScreenSpot Desktop and ScreenSpot-Pro)

Discussion. Across both splits, RL fine-tuning using our ACCUANNOTATE dataset improves AUC and reduces DTB. Gains are larger in relative accuracy on the more challenging ScreenSpot-Pro [3]. The above evaluations show that our methods of training ACCUANNOTATE-2B model are effective. However, our training is still limited by the size of our training dataset and we will discuss this in detail in Chapter 5.

Benchmark	Model / Epoch	AUC (%)	DTB (px)	Accuracy (%)
ScreenSpot [2] (334 Tasks)	ShowUI-2B	67.90	78.07	70.06
	ACCUANNOTATE-2B	**71.50**	**70.99**	**72.16**
	Δ	**+5.29%**	**-9.08%**	**+3.00%**
ScreenSpot-Pro [3] (1581 Tasks)	ShowUI-2B	8.88	635.15	13.79
	ACCUANNOTATE-2B	**10.39**	**633.58**	**15.62**
	Δ	**+16.95%**	**-0.25%**	**+13.27%**

TABLE 4.4: ACCUANNOTATE's Performance in Benchmarks

4.10 Robustness and Sensitivity

Decoding. We sweep decoding with $T \in \{0.3, 0.5, 0.7\}$, top-$k \in \{0, 20, 50\}$, top-$p \in \{0.0, 0.9\}$ (where $k = 0$ or $p = 0$ reduces to greedy). Report Success and Invalid.

Visual token budgets. We vary `min_visual_tokens` and `max_visual_tokens` within practical GPU limits and observe trade-offs between detail preservation and memory/runtime.

Resolution generalisation. Evaluate at original resolution and uniform resizes (e.g., long side $\{720, 1080, 1440\}$). Report Success/L2. This probes scale sensitivity without DOM access.

4.11 Error Analysis

We annotate a random sample of failures and group them into: (i) small clickable targets (icons/toggles), (ii) ambiguous linguistic references (multiple similar widgets), (iii) text-only cues requiring OCR-like reading, (iv) layout confusions (dense toolbars), and (v) parser failures.

CHAPTER 5

Discussion

In this chapter, we first summarise the principal results, then analyse the mechanisms by which the proposed methods yield improvements, assess robustness and generalisation, examine supervision quality and curation trade-offs, and finally discuss limitations, practical implications, and directions for future work.

5.1 Principal Findings

Across desktop-focused ScreenSpot splits and the more challenging ScreenSpot-Pro benchmark, the RL fine-tuned model (ACCUANNOTATE-2B) surpasses the frozen backbone (**ShowUI-2B**) under identical prompts, parsers, and decoding rules. In particular, Table 4.4 shows a consistent increase in cumulative hit-rate AUC and a reduction in distance-to-box (DTB), with a larger relative AUC gain on ScreenSpot-Pro. Because the evaluation protocol holds constant decoding (greedy by default), pre- and post-processing, and hardware settings, these improvements can be attributed to the combined effect of the dataset produced by ACCUANNOTATE and the RL optimisation applied to the compact VLM.

5.2 Mechanisms Underpinning the Gains

The observed improvements align with the intended roles of the training components introduced in Chapter 3. First, the distance-aware reward provides a dense learning signal that penalises off-centre predictions even when the click falls inside the ground-truth box. This pressure reduces DTB and, by shifting probability mass toward box centroids, indirectly converts borderline near-misses into hits, thereby increasing AUC. Second, the success-radius schedule (τ) acts as a curriculum: early permissiveness accelerates learning, while later tightening promotes precise localisation without destabilising decoding. Third, the EMA baseline and entropy regularisation reduce gradient variance and discourage premature

collapse, which is reflected qualitatively in lower invalid-format rates and smoother training dynamics (cf. ablation tendencies in Table 4.3). Finally, the greedy warm-up and safety cooldown suppress transient failure modes (e.g., malformed coordinate strings and high-entropy "meltdowns") that otherwise degrade reliability in coordinate-emitting policies.

5.3 Quality of Supervision and Curation Trade-offs

The credibility of training signals is supported by the high pipeline–human agreement. On $N{=}1{,}740$ tasks, ACCUANNOTATE achieves 98.77% accuracy with a narrow 95% CI of ± 0.27 percentage points under the click-inside-box criterion (Table 4.1). This level of agreement reduces the likelihood that gains stem from label noise artefacts and strengthens the causal link between supervision quality and downstream improvements.

A practical advantage of ACCUANNOTATE is its controllable detail level, which provides different levels of detail that different trainings need. In our setting, the Medium preset offered a favourable balance for large-scale exports, whereas High detail proved valuable when constructing targeted validation slices that stress fine-grained elements. Two design choices further underpin label fidelity: (i) the use of absolute-pixel geometry during curation (normalisation occurs only at export), which avoids rounding artefacts during training; and (ii) hint-constrained selection with post-hoc snapping, which curbs hallucination and enforces strict geometric consistency with detector hypotheses.

5.4 Robustness and Generalisation

Sensitivity analyses indicate that greedy decoding yields the most reliable single-shot behaviour for desktop grounding, consistent with our objective of producing a single accurate click. While stochastic decoding (e.g., top-k/top-p sampling) may be beneficial for interactive agents that can attempt multiple actions, it tends to increase invalid formats without improving expected DTB in the single-attempt regime. Visual token budget sweeps reveal diminishing returns once the local neighbourhood of the target element is adequately represented; beyond that point, larger budgets primarily increase memory and runtime. Resolution sweeps suggest encouraging transfer to moderate rescaling and unseen applications; however, given the predominance of desktop screenshots in training, we regard the current results as largely on-distribution to ScreenSpot's desktop slice and recommend broader cross-app validation in future work.

5.5 Practical Implications

The findings translate into several design recommendations for pixel-level GUIs grounded in desktop environments. During data curation, absolute-pixel labels should be preferred, with normalisation deferred to export. Constraining annotation to detector hints, interleaving context crops, and snapping outputs to hints reduces geometric drift and label variance. During optimisation, distance-shaped rewards combined with a tightening success radius provide a stable pathway from coarse to precise localisation. For evaluation and deployment in single-click tasks, greedy decoding is an effective default. Where retry budgets exist, limited stochasticity may be explored, but format guards should accompany it. Finally, checkpoint selection should consider multiple metrics (e.g., AUC and DTB) to avoid over-optimising for a single behavioural axis; Pareto screening is a natural choice in this setting.

5.6 Conclusion

In summary, automatically curated, hint-constrained, pixel-accurate annotations produced by ACCUANNOTATE provide reliable supervision for reinforcement-learning fine-tuning of a compact VLM. The distance-aware objective, stability heuristics, and curriculum over the success radius jointly improve both hit conversion and localisation precision on desktop GUI benchmarks. While the present evidence is strongest on ScreenSpot's desktop distribution, the methodology is general and suggests practical guidance for building accurate and efficient grounding systems in desktop environments.

CHAPTER 6

Threats to Validity

This chapter discusses potential threats to the validity of our findings and the steps taken (or planned) to mitigate them. We organise the discussion along *internal*, *external*, *construct*, and *conclusion* validity.

6.1 Internal Validity

Data leakage and near-duplicates. Although we enforce strictly held-out evaluation (no screenshot in training exports appears in the test splits; Chapter 4), leakage can still occur through near-duplicate screens (minor UI state changes, identical windows with different backgrounds, or templated app layouts). Such correlations can inflate apparent generalisation.

Mitigations. We deduplicate training exports before RL using perceptual hashing and filename/metadata heuristics, and we keep evaluation assets under separate export runs. As a planned safeguard, we will add near-duplicate filtering on the *union* of train/val/test with pHash clustering and app/window-title bucketing, then re-run the main evaluation on the pruned sets.

Training–evaluation coupling. All models share the same prompt, parser, clamping, and greedy decoding at evaluation time. While this controls variance, it risks coupling improvements to parser or prompt-specific behaviour learned during RL.

Mitigations. We hold pre/post-processing constant across baselines, reducing differential bias. To probe coupling, we schedule robustness checks with prompt paraphrases and a stricter parser variant (e.g., by tightening regex tolerances). A material drop under paraphrase would indicate prompt overfitting.

RL instability and seed sensitivity. Policy-gradient training is stochastic; gains might reflect favourable random seeds or transient plateaus.

Mitigations. We use fixed seeds for selection, batch size $= 1$ inference, and greedy decoding for determinism. The ablation script (in progress) aggregates across multiple seeds and reports seed-wise dispersion to separate algorithmic effects from stochasticity.

Reward/heuristic confounds. Heuristics such as warm-up, success-radius curriculum τ, safety cooldown, and distance-shaped rewards may reduce invalid formats or shift decoding entropy without truly improving grounding.

Mitigations. The ablation plan isolates each component with identical evaluation settings. We additionally report DTB alongside Success to distinguish formatting reliability from localisation accuracy.

6.2 External Validity

Benchmark and platform coverage. Results are strongest on desktop-focused ScreenSpot splits and ScreenSpot-Pro (Section 4.9). Transfer to mobile UIs, touch-first layouts, or web-heavy benchmarks may be weaker due to differences in widget scale, aspect ratios, and interaction affordances.

Mitigations. We report both desktop and the more diverse ScreenSpot-Pro and plan cross-benchmark tests (e.g., app diversity slices) and cross-resolution sweeps. Future work will include mobile-oriented corpora and dynamic UI states.

UI diversity: DPI, themes, i18n. DPI scaling, dark mode, localisation (non-Latin scripts, RTL), and accessibility settings can change visual salience and text rendering.

Mitigations. Resolution generalisation experiments vary the long side (e.g., $\{720, 1080, 1440\}$) and we plan language/contrast stress slices. Including OCR-oriented hints and multilingual text detectors in curation is a straightforward extension.

Interaction regime mismatch. Our evaluation uses single-shot clicking with greedy decoding; real agents may benefit from retries or exploratory sampling.

Mitigations. We present decoding sweeps and invalid-rate tracking; for deployment scenarios with attempt budgets, we recommend small-k/top-p sampling with format guards and will report success under fixed retry counts in follow-up experiments.

6.3 Construct Validity

Metric adequacy. Click-inside-box *Success* is coarse: a near-edge hit and a centred click are both counted as correct. DTB is absolute-pixel distance and is sensitive to scale; AUC (cumulative hit-rate) integrates over radii but does not model element size. Invalid-format rate mixes syntax reliability with semantics.

Mitigations. We report Success *and* DTB jointly, and use AUC as a complementary summary. Planned analyses include size-normalised distance (e.g., DTB divided by box diagonal) and radius-conditioned success curves to better capture precision. Invalid is always interpreted alongside DTB/Success.

Label correctness and independence. Pipeline–human agreement is high (98.77% $\pm$ 0.27 pp) under the same click-inside-box criterion, but this check uses a single rater and the same decision rule as the benchmark, which may hide systematic biases (e.g., consistent off-centre snapping). Hint-constrained labeling can inherit detector blind spots (low-contrast icons, text-only widgets).

Mitigations. We will conduct a dual-annotation study with adjudication on a stratified sample (small icons, text-only controls, ambiguous references), report inter-rater reliability, and add a small unhinted gold set to quantify hint coverage. Detector diversity (ensembling text/layout/control detectors) and post-hoc mining of false negatives are planned.

Success-radius curriculum and AUC. Training tightens τ over time; if evaluation summaries emphasise small radii, improvements may partly reflect alignment between the curriculum and the summary metric.

Mitigations. We report the full cumulative hit curve and DTB, and we will include fixed-radius checkpoints (no curriculum) in ablations to assess dependence.

6.4 Conclusion Validity

Statistical power and dependence. Speed caps on the desktop subset reduce N, lowering power for small effects. UI screens from the same application are not fully independent, which can narrow bootstrap intervals.

Mitigations. We use item-wise paired bootstraps for deltas and will add cluster-aware resampling at the app/window level. ScreenSpot-Pro's larger N complements the desktop subset for higher-power estimates.

Multiple comparisons and selection bias. Selecting the best checkpoint on a desktop subset and then reporting on related test data risks optimistic estimates if the selection signal correlates with the report set.

Mitigations. Selection uses a fixed subset and a metric not directly optimised by the loss; we will add a disjoint *report-only* slice and/or perform nested selection to avoid peeking. Where many ablations are compared, we will prefer effect sizes with CIs over p-values and avoid post-hoc thresholding.

Attribution without completed ablations. As the full ablation study is pending, causal attributions to specific components (distance term, EMA baseline, entropy, warm-up, safety cooldown, τ schedule) remain provisional.

Mitigations. The sequential ablation plan evaluates one toggle at a time, aggregates across seeds, and reports both central tendency and dispersion. Until those results are in, we treat component-level claims as hypotheses consistent with observed trends rather than definitive causal statements.

6.5 Summary

We reduce *internal* threats by controlling evaluation tooling across models, enforcing held-out splits, and planning duplicate-aware pruning; *external* threats by testing larger and more diverse splits and performing resolution/language stress checks; *construct* threats by pairing Success with DTB, auditing labels beyond hint constraints, and exploring size-normalised metrics; and *conclusion* threats by paired (and planned cluster-aware) bootstraps, seed aggregation, and nested selection. Residual risks primarily stem from dataset scope (desktop skew), hint-driven curation, and pending ablations; these are made explicit alongside mitigations and will be revisited once the ablation study is complete.

CHAPTER 7

Limitations and Future Work

This chapter summarises the key limitations of our current system and outlines concrete directions for future work. For consistency, we use *GUI* to refer to desktop graphical user interfaces.

7.1 Limitations

7.1.1 Dependence on OmniParser for Element Discovery

Our data-collection pipeline relies on **OmniParser–v2** for discovering candidate GUI elements before annotation. In practice, *OmniParser does not detect all on-screen UI elements*; it typically returns only a *subset* of widgets (often larger, text-bearing, or high-contrast controls). As a result:

- The **recall of dataset collection is upper-bounded** by OmniParser: false negatives directly translate into *missing annotations*, limiting coverage.
- False positives or loose boxes introduce **noisy pseudo-ground-truth**, which can degrade reward signals and learning stability.
- Certain categories are under-detected, including: custom/canvas-drawn widgets, icon-only buttons, ephemeral overlays (tooltips, context menus), system pop-ups, and elements within nested/scrolling containers.
- Visual/domain factors (dark mode, high-DPI scaling, theme customisation, motion/animation, transparency) further reduce detector robustness.

Overall, the effectiveness of ACCUANNOTATE is **largely limited by OmniParser's element-detection performance**. Improving discovery is therefore a first-order priority (see Section 7.2.1).

7.1.2 Annotation Quality and Noise

Although ACCUANNOTATE enforces format validity, pixel-accurate localisation is not guaranteed for every instance:

- OCR jitter and bounding-box imprecision can shift click targets by a few pixels.
- Dynamic content (hover states, animations) and non-determinism across render cycles introduce label variance.
- Version drift in upstream tools (detector/ocr libraries) can change coordinates over time, affecting reproducibility.

7.1.3 Coverage and Dataset Bias

Our collected data skews toward desktop applications and a limited set of OS/app stacks. This yields:

- Reduced generalisability to mobile/tablet/web UIs and niche enterprise software.
- Locale and accessibility bias (language, fonts, screen scaling, colour schemes) under-represented in training.

7.1.4 RL Fine-Tuning Constraints

Our RL runs target compact models with modest compute. Practical constraints include:

- **Seed variance** and short training horizons (few epochs) limit statistical confidence.
- Sensitivity to entropy and reward coefficients can affect exploration, whilst KL provided a modest accuracy lift but was not required for parse stability under greedy evaluation in our ablation (Section 4.8).
- Single-turn grounding may miss opportunities for multiturn refinement (coarse-to-fine localisation, clarification turns).

7.1.5 Reward Design

Our distance and format based rewards are simple and efficient, but:

- Distance-only shaping can over-reward near-misses just outside a box.

- Absence of region-overlap terms (e.g., IoU) under-penalises tangential predictions.
- Format compliance is binary and may not reflect semantic correctness.
- Our reward's success term uses a radius around the box centre, whereas evaluation success requires box containment (Section 4.4). On elongated or off-centre boxes these can diverge, which we discuss empirically in Section 4.8.

7.1.6 Evaluation Protocol

We primarily evaluate on desktop-focused ScreenSpot splits. Limitations include:

- Metrics (click-inside-box success, AUC-hit, distance-to-box) are proxies and may not fully capture agent utility.
- Greedy decoding underestimates performance of calibrated sampling or reranking strategies.
- Lack of end-to-end task evaluation (multi-step GUI tasks, OSWorld-like suites) limits external validity.
- Train–eval decoder mismatch: we use a regex/FSM-gated decoder during training but evaluate with greedy decoding for determinism (Section 3.2.6), which may understate benefits of constrained decoding or sampling with reranking.

7.1.7 System and Engineering Constraints

Constraints that affect throughput and reproducibility:

- Compute/memory budgets necessitate 8-bit loading and small batch sizes.
- Pipeline throughput (screenshotting, detection, OCR, I/O) can bottleneck large-scale collection.
- Version pinning and environment drift can impact exact reproducibility of coordinates and rewards.

7.1.8 Ethical, Legal, and Privacy Considerations

GUI screenshots may contain sensitive information. While our pipeline supports redaction and controlled collection, broader deployment must consider:

- Consent and data-governance for captured content.

- License compliance for software/assets depicted.
- On-device processing and PII minimisation where feasible.

7.2 Future Work

7.2.1 Stronger Element Discovery and Coverage

- **Improve/extend OmniParser**: fine-tune on GUI-specific corpora, add dark-mode and high-DPI variants, and calibrate confidence thresholds per widget type.
- **Detector ensembles**: combine general detectors with UI-specific heuristics, OCR cues, and small region-proposal models to raise recall.
- **Accessibility integration**: fuse accessibility trees/role metadata (when available) with visual detections for higher coverage and cleaner semantics.
- **Temporal cues**: stabilise boxes across frames to better capture ephemeral elements (menus, tooltips, toasts).

7.2.2 Human- and Model-in-the-Loop Data Curation

- **Active learning**: prioritise samples where detectors disagree or model uncertainty is high.
- **Self-training**: use high-confidence model predictions to expand labels, with periodic human spot checks.
- **Error mining**: systematically harvest near-misses and hard negatives to harden the reward and the policy.

7.2.3 Richer Annotations

- Move beyond point/box to *segmentation masks*, widget hierarchies, and state attributes (enabled/disabled, focus, visibility).
- Record interaction affordances and z-order to reflect real clickability.

7.2.4 Data Diversification and Synthesis

- Expand to mobile/tablet/web UIs; diversify OSes, locales, themes, and accessibility settings.

- Generate synthetic UIs (programmatic HTML/CSS renderers, design-tool exports) with controllable layouts and labels for coverage.

7.2.5 Learning Algorithm Improvements

- Explore *multi-turn* RL for coarse-to-fine localisation and clarification.
- Compare GRPO/DPO/IPO and off-policy critics to address sample inefficiency and instability.
- Add **adaptive KL/entropy** schedules and curriculum strategies.

7.2.6 Reward and Decoding Enhancements

- Incorporate overlap-aware terms (e.g., IoU, soft masks), distance robustification, and calibrated format rewards.
- Use constrained decoding to enforce output schema; evaluate top-k sampling with reranking by a verifier.

7.2.7 Evaluation Expansion

- Add task-oriented suites (e.g., multi-step GUI benchmarks) and human-in-the-loop usability checks.
- Report latency/throughput and robustness to theme/scaling changes.
- Run longer training horizons and *more seeds* to quantify variance.

7.2.8 System Engineering and Reproducibility

- Strengthen containerisation, version pinning, and dataset cards with provenance.
- Provide an interactive annotation QA tool and public scripts for end-to-end replication.

7.2.9 Responsible Release

- Expand redaction, consent flows, and on-device processing options.
- Clarify dataset licensing and permissible use; consider a research-only license where appropriate.

7.3 Summary

The dominant limitation of our pipeline is its **dependence on OmniParser for element discovery** (as stated in Section 7.1.1), which caps collection recall and injects noise. Complementary limits arise from annotation noise, dataset bias, RL instability, and evaluation scope. Our future work prioritises raising detection recall/precision, broadening data diversity, strengthening learning and rewards, and expanding evaluation toward end-to-end agent utility.

CHAPTER 8

Conclusion

This thesis set out to investigate whether automatically curated, pixel-accurate supervision can enable effective reinforcement learning (RL) fine-tuning of a compact vision–language model (VLM) for desktop GUI grounding, and to characterise the conditions under which such improvements are reliable and practically applicable.

8.1 Summary of Contributions

(1) **AccuAnnotate pipeline.** We introduced a hint-constrained annotation pipeline that produces high-fidelity point-in-box labels from screenshots with controllable detail presets (Low/Medium/High) for cost–coverage trade-offs. A human study on $N{=}1{,}740$ tasks found **98.77%** agreement with an estimated 95% CI of ± 0.27 pp under the benchmark criterion, supporting the credibility of the supervision signal.

(2) **RL fine-tuning for precise localisation.** We designed a distance-aware objective with an EMA baseline, entropy regularisation, a success-radius curriculum (τ), greedy warm-up, and a safety cooldown to stabilise coordinate-emitting policies.

(3) **Evaluation harness and metrics.** We reported click-inside-box *Success*, L2 distance-to-point, invalid-format rate, AUC (cumulative hit rate), and DTB (distance-to-box), with item-wise 95% bootstrap intervals and paired deltas for comparative claims, under a deterministic evaluation protocol (shared prompt, parser, and greedy decoding).

8.2 Empirical Findings

Across the desktop subset of *ScreenSpot* and the more diverse *ScreenSpot-Pro*, the RL fine-tuned model (ACCUANNOTATE-2B) consistently surpassed the frozen backbone (SHOWUI-2B) under matched decoding and parsing (Chapter 4). On the desktop slice, AUC increased from **67.90** to **71.50** (+5.29%),

with DTB reduced from **78.07** to **70.99** px (-9.08%). On ScreenSpot-Pro, AUC rose from **8.88** to **10.39** (+16.95%), with a modest DTB improvement. These gains align with the intended mechanisms: dense distance shaping shifts probability mass toward box centroids, the τ-schedule mediates a curriculum from coarse to precise localisation, and stability heuristics curb invalid outputs. Robustness sweeps indicate greedy decoding is a strong default for single-attempt grounding; stochastic sampling can raise invalid rates without commensurate localisation benefits in this regime (Chapter 5).

8.3 Answer to the Research Question

Previously in section Section 1.3, we have defined our research question as:

> How can we automatically produce high-fidelity, pixel-accurate annotations for desktop GUI grounding, and to what extent do datasets generated by this pipeline enable reinforcement learning fine-tuning of a compact VLM to improve grounding performance in desktop environments?

Yes. Automatically generated, hint-constrained, pixel-level annotations are accurate enough to supervise RL fine-tuning of a compact VLM. In controlled evaluations on desktop GUIs, this leads to measurable gains in grounding—higher hit rates (AUC/Success) and tighter localisation (lower DTB). The approach is practical because screenshots scale easily, and a distance-shaped reward with simple stabilising heuristics provides a reliable training signal. These gains are strongest for desktop data that closely resembles our training exports. Wider generalisation will require further validation.

8.4 Limitations

Several factors bound the scope of these conclusions (Chapters 6 and 7). Most notably: (i) dataset scale and cost constrained the richness of contextual descriptions; (ii) sampling imbalance (e.g., frequent head icons) likely under-serves the long tail of rare widgets; (iii) the 2B-parameter backbone limits OCR fidelity and long-range context; (iv) hint-constrained labels may import detector blind spots; (v) evaluation targets single-shot clicking on desktop benchmarks, leaving interactive, multi-step behaviour and mobile/touch-first UIs underexplored; and (vi) metric coarseness (Success vs. centring, size-normalisation) leaves room for more geometry-sensitive reporting. Pending full ablations, causal attributions to individual training components remain provisional.

8.5 Implications and Guidance

Practically, our results suggest a set of effective defaults for pixel-level grounding on desktop: absolute-pixel labels (normalise only at export), hint-constrained selection with post-hoc snapping, distance-shaped rewards with a tightening success radius, and greedy decoding with strict output guards at evaluation. For checkpoint selection, consider multi-metric screening (AUC and DTB) rather than a single scalar to avoid over-optimising one behavioural axis.

8.6 Future Research

Building on the above, several directions appear most impactful:

- **Targeted data scaling.** Active learning and stratified, class-aware sampling to densify rare widgets and ambiguous contexts without linear growth in API cost; selective use of *High*-detail exports.
- **Richer supervision.** Context-aware descriptions where ambiguity is detected; detector diversity (text/layout/control) with competitive re-snapping and audits on an unhinted gold slice.
- **Objectives and metrics.** Size-aware, Gaussian-like point-to-box rewards; size-normalised DTB and radius-conditioned success curves to better reflect perceived precision.
- **Model scaling with efficiency.** Moderate increases in capacity (4–9B, MoE) combined with parameter-efficient RL (LoRA/DoRA) and post-training quantisation; teacher–student distillation to keep runtime costs low.
- **Generalisation and interaction.** Cross-benchmark, cross-resolution, and multilingual/accessibility stress tests; evaluation under fixed retry budgets and simple multi-step programs to reflect real workflows.
- **Governance and reproducibility.** Release of evaluation harnesses, seeds, and curation logs; privacy-preserving redaction and license-aware filtering for broader sharing.

8.7 Closing Remarks

This thesis demonstrates that careful data curation, combined with simple, principled RL shaping, can make a compact VLM meaningfully better at pixel-level grounding on desktop GUIs. The approach is practical, sample-efficient, and provides clear levers (reward shaping, decoding policy) for further

improvement. With targeted scaling, richer supervision, and broader evaluation, these ideas can extend beyond desktop screenshots toward robust, interactive GUI agents capable of reliable action in diverse real-world environments.

APPENDIX A

AccuAnnotate Case Analysis

In this appendix, we demonstrate how researchers can use our ACCUANNOTATE pipeline to prepare training data for UI grounding tasks. ACCUANNOTATE comprises five UI areas that together support ingestion, organisation, annotation, verification, and export of datasets.

A.1 Workspace

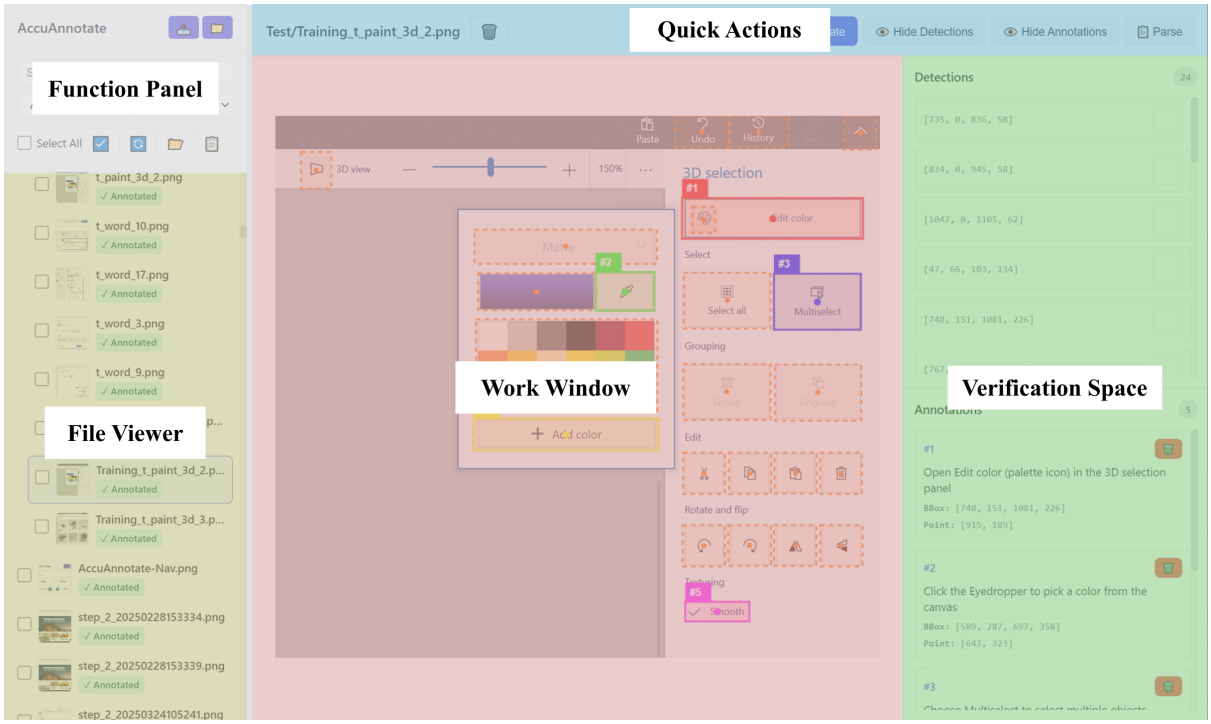

FIGURE A.1: ACCUANNOTATE's Workspace

As shown in Fig. A.1, the workspace includes: - Function Panel (global controls) - File Viewer (image list with filters) - Work Window (image canvas and overlays) - Verification Space (Detections and Annotations) - Quick Actions (per-image operations)

A.2 Function Panel

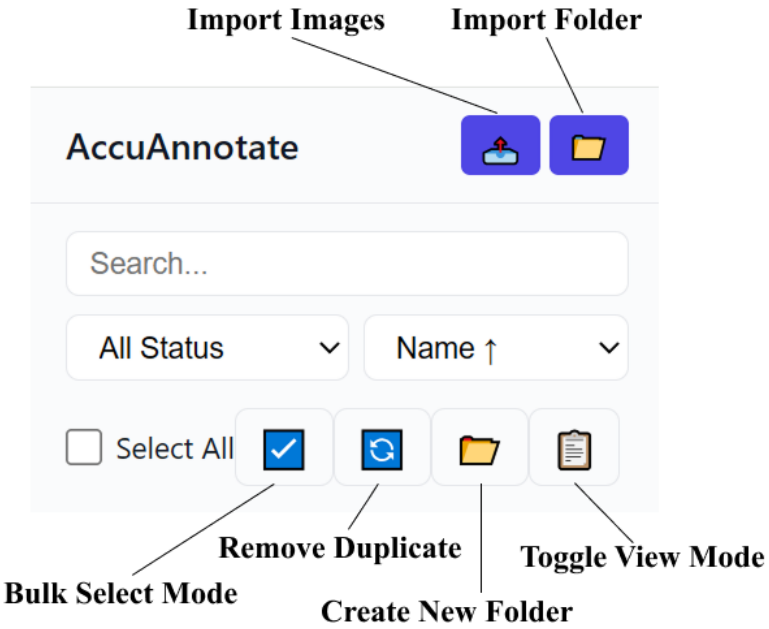

FIGURE A.2: ACCUANNOTATE's Function Panel

Fig. A.2 shows the global controls for data management and navigation: importing files/folders, searching, sorting and filtering, deduplication, bulk selection, folder creation, and view mode toggling. When items are selected, bulk actions (Annotate, Delete, Export, Clear) appear contextually.

Controls

Import Images: Opens the upload modal for drag/drop or multifile selection. Supported types: JPG, JPEG, PNG, GIF, BMP. Uploaded files are saved under `data/images/` and immediately indexed into the viewer.

Import Folder: Imports a full directory using browser folder selection. The original subfolder structure is preserved (via uploaded `relative_path`), allowing downstream folderbased organisation.

Search / Filter / Sort: - Search: substring match on filename (live filtering). - Filter: *All*, *Annotated*, or *Not Annotated*. - Sort: name (↑ / ↓) or status (↑ / ↓).

Bulk Select Mode: Toggles a selectionfirst workflow. Click to select; *Shift+Click* selects a range; *Ctrl/Cmd+Click* toggles individual items. A *Select All* checkbox mirrors the current filtered set. Selected items become draggable.

Remove Duplicate: Deduplicates by base filename across folders, keeping the most useful copy (prioritising annotated) and removing other duplicates and their annotations. Intended for quickly cleaning large crawls.

Create New Folder: Creates a folder under `data/images/` and registers it in the workspace so it appears in folder view and the move menu. Images can be moved into folders via drag/drop or the image context menu.

Toggle View Mode: Switches between *List* view (flat) and *Folder* view (hierarchical). Folder view supports expand/collapse and shows empty folders so users can preorganise targets.

Bulk Actions

Bulk Annotate: Runs asynchronous batch annotation with live progress. Uses preprocessing hints and the VLM to generate annotations for all selected images.

Bulk Delete: Deletes selected images (and their annotations if present).

Export Dataset: Exports the selected set to a chosen format and split, returning a downloadable ZIP. Currently supports *ShowUI-Desktop*; new formats (e.g., COCO, YOLO) can be added via scripts.

Clear Selection: Clears the current selection set and hides bulk actions.

A.3 File Viewer

The File Viewer presents the image list with thumbnails, status badges (Annotated/Not annotated), and checkboxes for selection. Thumbnails are lazy-loaded for scalability. In *Folder* view, images are grouped by directories with expand/collapse, a context menu for folder operations (delete, rename placeholder), and drag/drop of selected images into target folders. Rightclicking an image opens an image context menu (Delete, Move To).

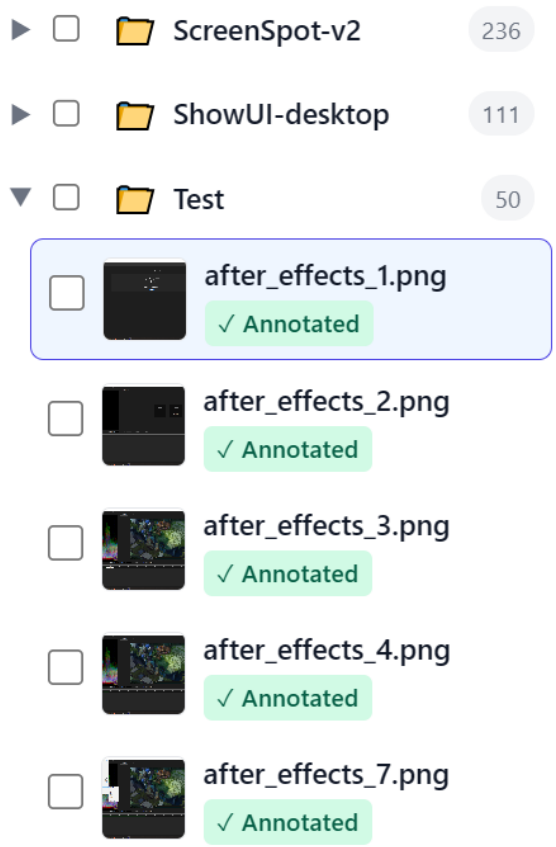

FIGURE A.3: ACCUANNOTATE's File Viewer

A.4 Work Window

The Work Window centres the selected image and overlays visual elements on a canvas. Users can toggle: - *Detections*: preprocessed UI candidates from the detector (dashed orange boxes and points). - *Annotations*: final labels returned by the VLM (colorcoded boxes, centre points, numeric tags).

Overlays are scaled to the displayed size and remain consistent on resize. The header area displays the current filename and provides Quick Actions (below).

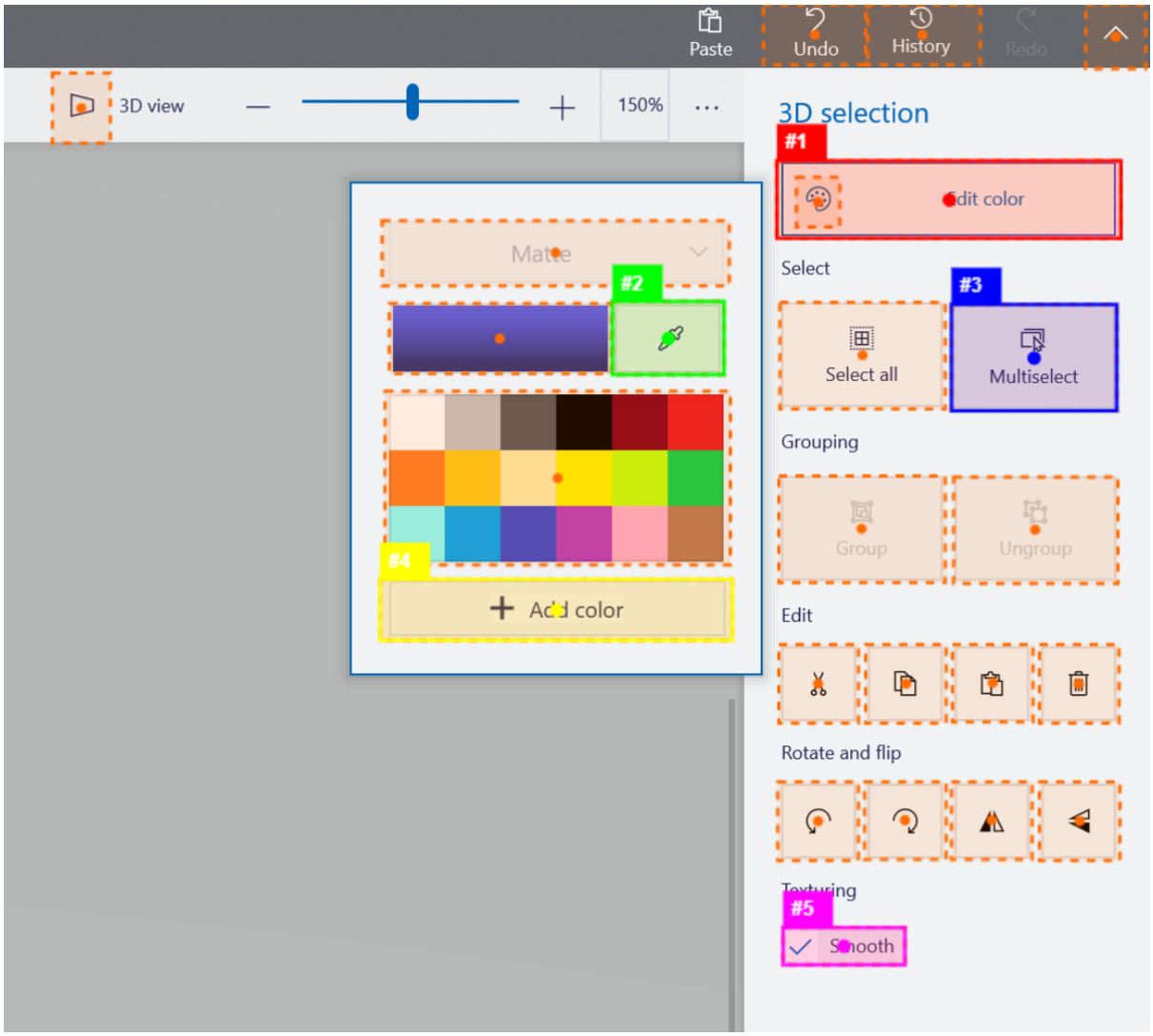

FIGURE A.4: ACCUANNOTATE's Work Window with visualised bounding boxes

A.5 Verification Space

The right panel contains two widgets: - **Detections**: candidates from the preprocessing step (bounding boxes and points). Users can prune false positives before annotation. - **Annotations**: the final elements (instruction, `bbox`, `point`). Selecting an element highlights it in the canvas; perelement deletion is supported for quick correction.

Counts for detections and elements are displayed at the panel headers.

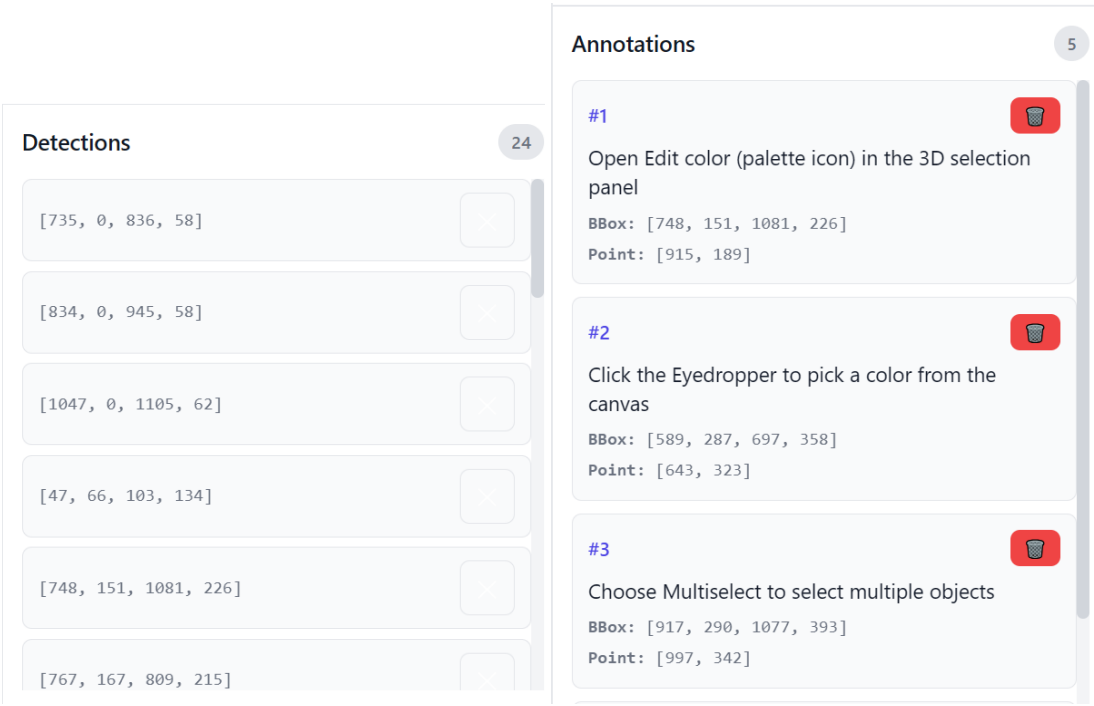

FIGURE A.5: ACCUANNOTATE's Verification Space with Detections and Annotations.

A.6 Quick Actions

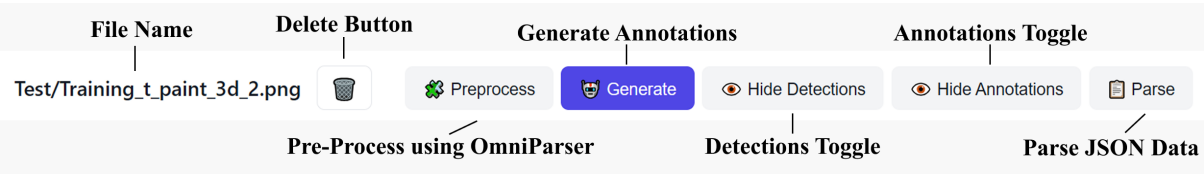

FIGURE A.6: Quick Actions: per-image operations and visualisation toggles.

Quick Actions appear in the Work Window header and include:

Preprocess: Runs OmniParserstyle detection to propose candidate UI elements (nonLLM).

Generate: Sends the image (plus hints) to the VLM and returns annotations.

Detections: Toggles the display of preprocessed candidates on the canvas.

Annotations: Toggles visualisation of annotated elements (colorcoded overlays).

Parse: Opens the JSON paste modal to manually apply userprovided annotations after validation.

Delete Image: Removes the current image (and its annotation if present).

A.7 Annotation Format and Coordinates

ACCUANNOTATE saves annotations as JSON with *absolute pixel* coordinates:

```
{
  "img_size": [width, height],
  "element": [
    {
      "instruction": "<120 chars>",
      "bbox": [x1, y1, x2, y2],
      "point": [cx, cy]
    }
  ]
}
```

- `bbox` is $[x_1, y_1, x_2, y_2]$ in pixels; `point` is $[c_x, c_y]$ in pixels. - The canvas scales these values to the displayed size for faithful visualisation. - The system remains backward compatible with older normalised annotations, autodetecting their format.

A.8 Exporting Datasets

Using the Function Panel's bulk actions, selected images can be exported via the *Export Dataset* modal: - Choose a dataset *Format* (currently *ShowUIDesktop*; extensible to *COCO*, *YOLO* via scripts). - Specify a *Split* name (e.g., *train*, *val*, *test*). - The server performs conversion and packaging and returns a ZIP for download.

A.9 Typical Workflow

A typical dataset creation run proceeds as follows:

(1) **Ingest** images via *Import Images* or *Import Folder*. Use *Search*, *Filter*, and *Sort* to find candidate subsets. Run *Remove Duplicate* to clean duplicates.

(2) **Organise** with *Folder* view: create folders, then drag/drop or *Move To* selected images into target folders.

(3) **Preprocess** a sample image to inspect candidate *Detections*. Remove spurious candidates if desired.

(4) **Annotate** the image with *Generate*, or *Parse* to paste a curated JSON. Iterate if necessary (delete incorrect elements).

(5) **Verify** in the *Verification Space*: inspect instructions, boxes, and points; toggle overlays to confirm placement and completeness.

(6) **Scale up** with *Bulk Annotate* on a selected set or folder, monitoring live progress.

(7) **Export** the curated subset with *Export Dataset*, selecting format and split, then download the ZIP for training.

References

[1] C. Lu, Z. Ge, G. Qin *et al.*, "Omniparser: A unified framework for text spotting, key information extraction and table recognition," *arXiv preprint arXiv:2403.19128*, 2024. [Online]. Available: https://arxiv.org/abs/2403.19128

[2] K. Cheng, Q. Sun, Y. Chu, F. Xu, L. YanTao, J. Zhang, and Z. Wu, "Seeclick: Harnessing GUI grounding for advanced visual GUI agents," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Long Papers)*. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 9313–9332, introduces the *ScreenSpot* benchmark. [Online]. Available: https://aclanthology.org/2024.acl-long.505/

[3] K. Li, Z. Meng, H. Lin, Z. Luo, Y. Tian, J. Ma, Z. Huang, and T.-S. Chua, "Screenspot-pro: Gui grounding for professional high-resolution computer use," *arXiv preprint arXiv:2504.07981*, 2025. [Online]. Available: https://arxiv.org/abs/2504.07981

[4] H. Li, J. Chen, J. Su, Y. Chen, Q. Li, and Z. Zhang, "Autogui: Scaling gui grounding with automatic functionality annotations from llms," *arXiv preprint arXiv:2502.01977*, 2025, accepted to ACL 2025 Main. [Online]. Available: https://arxiv.org/abs/2502.01977

[5] A. Memon, A. Nagarajan, and Q. Xie, "Automating regression testing for evolving gui software," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 17, no. 1, pp. 27–64, 2005.

[6] M. Grechanik, Q. Xie, and C. Fu, "Creating gui testing tools using accessibility technologies," in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2009, pp. 243–252.

[7] K. M. Conroy, M. Grechanik, and M. Hellige, "Automatic test generation from gui applications for testing web services," in *IEEE Software Engineering Notes*. IEEE, 2007.

[8] B. Wang, G. Li, X. Zhou, Z. Chen, T. Grossman, and Y. Li, "Screen2words: Automatic mobile UI summarization with multimodal learning," in *Proceedings of the 34th Annual ACM Symposium on User Interface Software and Technology (UIST '21)*. Association for Computing Machinery, 2021, pp. 498–510. [Online]. Available: https://www.dgp.toronto.edu/~bryanw/pdf/screen2words.pdf

[9] Y. Li, G. Li, L. He, J. Zheng, H. Li, and Z. Guan, "Widget captioning: Generating natural language description for mobile user interface elements," *arXiv preprint arXiv:2010.04295*, 2020, eMNLP 2020. [Online]. Available: https://arxiv.org/abs/2010.04295

[10] C. Bai, X. Zang, Y. Xu, S. Sunkara, A. Rastogi, J. Chen, and B. A. y Arcas, "Uibert: Learning generic multimodal representations for ui understanding," *arXiv preprint arXiv:2107.13731*, 2021,

iJCAI 2021. [Online]. Available: https://arxiv.org/abs/2107.13731

[11] Z. He, S. Sunkara, X. Zang, Y. Xu, L. Liu, N. Wichers, G. Schubiner, R. Lee, J. Chen, and B. A. y Arcas, "Actionbert: Leveraging user actions for semantic understanding of user interfaces," *arXiv preprint arXiv:2012.12350*, 2020, aAAI 2021. [Online]. Available: https://arxiv.org/abs/2012.12350

[12] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar, "Rico: A mobile app dataset for building data-driven design applications," in *Proceedings of the 30th ACM Symposium on User Interface Software and Technology (UIST)*, 2017.

[13] H. Furuta, K.-H. Lee, O. Nachum, Y. Matsuo, A. Faust, S. S. Gu, and I. Gur, "Multimodal web navigation with instruction-finetuned foundation models," in *ICLR 2024*, 2023. [Online]. Available: https://arxiv.org/abs/2305.11854

[14] X. Deng, Y. Gu, B. Zheng *et al.*, "Mind2web: Towards a generalist agent for the web," *arXiv preprint arXiv:2306.06070*, 2023. [Online]. Available: https://arxiv.org/abs/2306.06070

[15] T. Xie, D. Zhang, J. Chen, X. Li, S. Zhao, R. Cao, T. J. Hua, Z. Cheng, D. Shin, F. Lei, Y. Liu, Y. Xu, S. Zhou, S. Savarese, C. Xiong, V. Zhong, and T. Yu, "Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments," in *NeurIPS 2024 Datasets and Benchmarks*, 2024. [Online]. Available: https://arxiv.org/abs/2404.07972

[16] J. Yang, H. Zhang, F. Li, X. Zou, C. Li, and J. Gao, "Set-of-mark prompting unleashes extraordinary visual grounding in GPT-4V," *arXiv preprint arXiv:2310.11441*, 2023. [Online]. Available: https://arxiv.org/abs/2310.11441

[17] A. Yan, Z. Yang, W. Zhu, K. Lin, L. Li, J. Wang, J. Yang, Y. Zhong, J. McAuley, J. Gao, Z. Liu, and L. Wang, "Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation," *arXiv preprint arXiv:2311.07562*, 2023. [Online]. Available: https://arxiv.org/abs/2311.07562

[18] OpenAI, "GPT-4V(ision) system card," OpenAI, Tech. Rep., 2023. [Online]. Available: https://cdn.openai.com/papers/GPTV_System_Card.pdf

[19] K. Q. Lin, L. Li, D. Gao, Z. Yang, S. Wu, Z. Bai, W. Lei, L. Wang, and M. Z. Shou, "Showui: One vision-language-action model for gui visual agent," in *CVPR 2025*, 2024. [Online]. Available: https://arxiv.org/abs/2411.17465

[20] Z. Lu, Y. Chai, Y. Guo, X. Yin, L. Liu, H. Wang, H. Xiao, S. Ren, G. Xiong, and H. Li, "Ui-r1: Enhancing efficient action prediction of gui agents by reinforcement learning," *arXiv preprint arXiv:2503.21620*, 2025, introduces rule-based RL for GUI agents. [Online]. Available: https://arxiv.org/abs/2503.21620

[21] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," in *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021. [Online]. Available: https://proceedings.mlr.press/v139/radford21a.html

[22] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022. [Online]. Available:

https://arxiv.org/abs/2204.02311

[23] D. Driess, F. Xia, M. Sajjadi, C. Lynch, A. Toshev, P. Lambert *et al.*, "Palm-e: An embodied multimodal language model," *arXiv preprint arXiv:2303.03378*, 2023. [Online]. Available: https://arxiv.org/abs/2303.03378

[24] Z. Yang, L. Li, J. Wang, L. Zhang, L. Wang, and J. Luo, "The dawn of lmms: Preliminary explorations with GPT-4V(ision)," *arXiv preprint arXiv:2309.17421*, 2023. [Online]. Available: https://arxiv.org/abs/2309.17421

[25] Q. Guo, S. D. Mello, H. Yin, W. Byeon, K. C. Cheung, Y. Yu, P. Luo, and S. Liu, "Regiongpt: Towards region understanding vision language model," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 13 796–13 806. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2024/papers/Guo_RegionGPT_Towards_Region_Understanding_Vision_Language_Model_CVPR_2024_paper.pdf

[26] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations (ICLR)*, 2021. [Online]. Available: https://arxiv.org/abs/2010.11929

[27] G. Li and Y. Li, "Spotlight: Mobile ui understanding using vision-language models with a focus," in *International Conference on Learning Representations (ICLR)*, 2023. [Online]. Available: https://arxiv.org/abs/2209.14927

[28] G. Baechler, S. Sunkara, M. Wang, F. Zubach, H. Mansoor, V. Etter, V. Cărbune, J. Lin, J. Chen, and A. Sharma, "Screenai: A vision-language model for ui and infographics understanding," *arXiv preprint arXiv:2402.04615*, 2024, accepted to IJCAI 2024. [Online]. Available: https://arxiv.org/abs/2402.04615

[29] Y. Qian, Y. Lu, A. Hauptmann, and O. Riva, "Visual grounding for user interfaces," in *NAACL 2024 Industry Track*, 2024. [Online]. Available: https://aclanthology.org/2024.naacl-industry.9.pdf

[30] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," *arXiv preprint arXiv:2304.08485*, 2024, extended version of the 2023 preprint. [Online]. Available: https://arxiv.org/abs/2304.08485

[31] Y. Jiang, E. Schoop, A. Swearngin, and J. Nichols, "Iluvui: Instruction-tuned language–vision modeling of uis from machine conversations," *arXiv preprint arXiv:2310.04869*, 2023, to appear in a future IUI venue. [Online]. Available: https://arxiv.org/abs/2310.04869

[32] K. You, H. Zhang, E. Schoop, F. Weers, A. Swearngin, J. Nichols, Y. Yang, and Z. Gan, "Ferret-ui: Grounded mobile ui understanding with multimodal llms," in *European Conference on Computer Vision (ECCV) Workshops*, 2024. [Online]. Available: https://arxiv.org/abs/2404.05719

[33] H. You, H. Zhang, Z. Gan, X. Du, B. Zhang, Z. Wang, L. Cao, S.-F. Chang, and Y. Yang, "Ferret: Refer and ground anything anywhere at any granularity," *arXiv preprint arXiv:2310.07704*, 2023. [Online]. Available: https://arxiv.org/abs/2310.07704

[34] Z. Li, K. You, H. Zhang, D. Feng, H. Agrawal, X. Li, M. P. S. Moorthy, J. Nichols, Y. Yang, and Z. Gan, "Ferret-ui 2: Mastering universal user interface understanding across platforms,"

arXiv preprint arXiv:2410.18967, 2024, accepted to ICLR 2025 (Poster). [Online]. Available: https://arxiv.org/abs/2410.18967

[35] X. Chu, L. Qiao, X. Lin, S. Xu, Y. Yang, Y. Hu, F. Wei, X. Zhang, B. Zhang, X. Wei, and C. Shen, "Mobilevlm : A fast, strong and open vision language assistant for mobile devices," *arXiv preprint arXiv:2312.16886*, 2023. [Online]. Available: https://arxiv.org/abs/2312.16886

[36] G. Gemini Team, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," Google DeepMind, Tech. Rep., 2024. [Online]. Available: https://www.kapler.cz/wp-content/uploads/gemini_v1_5_report.pdf

[37] H. Wang, H. Zou, H. Song, J. Feng, J. Fang, J. Lu *et al.*, "Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning," *arXiv preprint arXiv:2509.02544*, 2025. [Online]. Available: https://arxiv.org/abs/2509.02544

[38] M. Xie, S. Feng, Z. Xing, C. Chen, and J. Fu, "Uied: A hybrid tool for gui element detection," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) Tool Demos*, 2020. [Online]. Available: https://sidongfeng.github.io/papers/uied.pdf

[39] S. Park, Y. Song, S. Lee, J. Kim, and J. Seo, "Leveraging multimodal LLM for inspirational user interface search," in *CHI Conference on Human Factors in Computing Systems (CHI '25)*, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3706598.3714213

[40] A. Burns, D. Arsan, S. Agrawal, R. Kumar, K. Saenko, and B. A. Plummer, "A dataset for interactive vision language navigation with unknown command feasibility," in *European Conference on Computer Vision (ECCV)*, 2022. [Online]. Available: https://github.com/aburns4/MoTIF

[41] Z. Hong *et al.*, "Multiui: Harnessing webpage uis for text-rich visual understanding," OpenReview preprint, 2024. [Online]. Available: https://openreview.net/forum?id=IIsTO4P3Ag

[42] Y. Gou *et al.*, "Universal visual grounding for GUI agents," *arXiv preprint arXiv:2410.05243*, 2024. [Online]. Available: https://arxiv.org/abs/2410.05243

[43] W. Hong, W. Wang, Q. Lv, J. Xu, W. Yu, J. Ji, Y. Wang, Z. Wang, Y. Dong, M. Ding, and J. Tang, "Cogagent: A visual language model for gui agents," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 14 281–14 290, introduces CCS400K with UI REG/REC (Box2DOM/DOM2Box) tasks via Playwright-rendered DOM/box pairs. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2024/papers/Hong_CogAgent_A_Visual_Language_Model_for_GUI_Agents_CVPR_2024_paper.pdf

[44] T. Beltramelli, "pix2code: Generating code from a graphical user interface screenshot," in *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS) Workshop*, 2017. [Online]. Available: https://arxiv.org/abs/1705.07962

[45] E. Z. Liu, K. Guu, P. Pasupat, T. Shi, and P. Liang, "Reinforcement learning on web interfaces using workflow-guided exploration," in *International Conference on Learning Representations (ICLR)*, 2018. [Online]. Available: https://arxiv.org/abs/1802.08802

[46] OpenAI, "Gpt-5," https://openai.com/index/introducing-gpt-5/, 2025, large language model.

[47] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992. [Online]. Available: https://doi.org/10.1007/BF00992696

[48] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," in *arXiv preprint arXiv:1707.06347*, 2017. [Online]. Available: https://arxiv.org/abs/1707.06347