TVAgent: A lightweight Vision-Language-Model for TV GUI Agent

Hoin Jung[*1,†], **Jinyang Liu**[*2,†], **Anirudh Rao**[3], **Hong-hoe Kim**[3], **Xiangyuan Zhao**[3], **Ashwin Chandra**[3], **Michel Sarkis**[3]

[1]Elmore Family School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907, USA
[2]Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115, USA
[3]Visual Display Intelligence Lab, Samsung Research America, Irvine, CA 92612, USA
[†]Work done during an internship at Samsung Research America

Abstract

Smart TVs are central to modern home entertainment, yet their interfaces remain cumbersome to navigate, especially for tasks like searching video content using on-screen keyboards. We introduce **TVAgent**, a lightweight vision–language model (VLM)–based system that enables reliable, real-time Smart TV interaction through keyword-based video search and seamless intra-page and inter-page navigation. To handle diverse and frequently changing app layouts, **TVAgent** integrates: (1) a fine-tuned lightweight VLM for robust and generalizable GUI parsing, trained with a synthetic data generation pipeline that produces realistic, annotated Smart TV layouts to enable scalable training and rapid domain adaptation, (2) a Multinomial–Dirichlet Modeling (MDM)–driven navigation module specialized for video content search, allowing adaptive traversal of heterogeneous virtual keyboards without manual layout mapping, and (3) a dynamic knowledge base for platform-aware adaptation, reducing redundant exploration and improving responsiveness over time. Across real apps, TVAgent achieves **97.7%** success on page-level navigation and **100%** on video search, with **83.0%** on intra-page content navigation; latency is sub-second throughout. TVAgent addresses pressing usability barriers in Smart TV navigation and demonstrates a clear path to near-term, on-device integration—offering significant potential for enhancing accessibility, efficiency, and personalization in home entertainment systems.

Introduction

Smart TVs are now a primary gateway for streaming media, yet their interfaces remain challenging to navigate efficiently—particularly for keyword-based video search. While agentic AI models for Graphical User Interfaces (GUIs) have advanced rapidly in mobile (Wang et al. 2025; Liu et al. 2025) and web domains (Zhang et al. 2025; Li et al. 2025), powered by large multimodal and vision–language–action models (Cui et al. 2024; Sapkota et al. 2025), Smart TV platforms have received far less attention despite their global ubiquity in daily entertainment.

Smart TV GUI automation faces distinctive challenges:

1. Heterogeneous and dynamic layouts: Streaming apps for movies, sports, and live TV each feature unique designs, navigation policies, and frequent content updates. These differences hinder the development of a single agent capable of reliable cross-app generalization.

2. Fundamentally different action space: In computers and mobile devices, actions often take the form of coordinate-based clicks or gestures. In contrast, Smart TVs rely on a small set of discrete directional commands (UP, DOWN, LEFT, RIGHT) plus selection. This simplicity forces the agent to generate long, context-dependent action sequences even for seemingly simple tasks like video search.

3. Resource limitations: On-device model inference must be lightweight to meet the computational and latency constraints of consumer hardware.

4. Need for a Universal Interface: While modern solutions like voice assistants or magic pointers exist, their functionality is often limited to specific applications or hardware ecosystems. They do not offer a universal control mechanism capable of operating across all apps on an OS. There remains a need for a lightweight, general-purpose agent that can understand and navigate *any* TV GUI, including those on unseen platforms, by operating at the OS level.

These combined challenges make keyword-based video search especially problematic: over 93% of users report frustration with navigating virtual keyboards (Khan et al. 2022; Alam, Khusro, and Khan 2019). Overcoming this barrier could significantly improve accessibility, efficiency, and personalization in Smart TV use.

To this end, we propose TVAgent, a lightweight, deployable Smart TV GUI agent designed to address these emerging usability challenges. Our main contributions are:

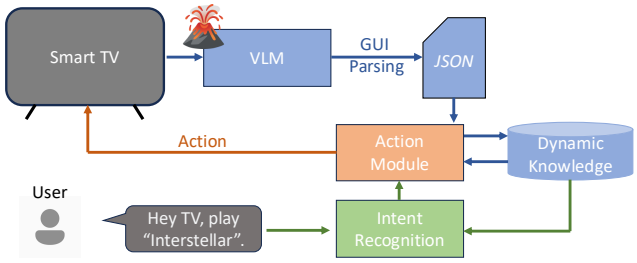


Figure 1: Overview of the proposed Smart TV GUI agent pipeline.

[*]These authors contributed equally.

1. **Lightweight VLM-based GUI parsing with synthetic data generation:** A fine-tuned, on-device–ready VLM for robust screen understanding and query interpretation across diverse apps, trained with a synthetic data generation pipeline that produces realistic, annotated Smart TV layouts for cost-effective training and rapid domain adaptation.
2. **Bayesian navigation for video search:** A Multinomial–Dirichlet Modeling (MDM)–based (Blei, Ng, and Jordan 2003; Harrison et al. 2020) navigation component that adaptively traverses heterogeneous virtual keyboards under layout uncertainty without requiring ground-truth layouts.
3. **Dynamic knowledge base:** An online clustering–driven module that retains app-specific interaction knowledge to minimize redundant exploration and parsing, improving responsiveness and user experience.
4. **Demonstrated readiness for deployment:** Experiments on multiple commercial Smart TV applications show that TVAgent achieves perfect navigation accuracy with sub-second latency, outperforming state-of-the-art vision–language–action models in robustness and generalization.

Related Works

GUI Understanding

GUI understanding methods aim to extract structured representations of user interface elements for downstream reasoning and action. Prior approaches fall into two broad categories.

The first category develops GUI-specialized models tailored for structured parsing. Examples include ScreenAI (Baechler et al. 2024) and Screen2Vec (Li et al. 2021), which learn semantic embeddings of GUI components and layouts, as well as GUI2DSVec (Yang and Li 2023) and V-Zen (Rahman et al. 2024), which target grounding accuracy and interface design quality. While effective for web and mobile platforms, these methods do not address the dynamic, heterogeneous layouts and virtual keyboard structures found in Smart TV interfaces.

The second category fine-tunes general-purpose vision–language models for GUI-related tasks. Works such as GUICourse (Chen et al. 2025), CogAgent (Hong et al. 2024), and Mp-GUI (Wang et al. 2025) adapt pretrained multimodal models for classification, detection, or parsing in GUI environments. These methods leverage the broad visual–textual understanding of large pretrained models but still require significant domain-specific data for high performance on unseen platforms.

Synthetic data generation has emerged as an effective way to address data scarcity in GUI tasks, enabling cost-effective training without extensive manual annotation (Peng et al. 2024; Maniyar et al. 2025). Existing datasets are primarily designed for computer and mobile interfaces. In contrast, our work introduces a Smart TV–specific synthetic data pipeline that includes layout randomization, diverse virtual keyboard designs, and complete interface context—enabling robust GUI parsing that directly supports downstream navigation.

Vision-Language Action Models for GUI

Vision–language–action (VLA) models directly map screen and text inputs to action sequences in a unified framework. UITARS (Qin et al. 2025), LearnAct (Liu et al. 2025), TongUI (Zhang et al. 2025), and ShowUI (Lin et al. 2025) have demonstrated strong performance for mobile and web interfaces by predicting actions end-to-end.

However, these models typically assume coordinate-based action spaces and do not explicitly address the discrete directional navigation found in Smart TV platforms, where even simple tasks like typing a movie title can require long, context-aware sequences. Their reliance on direct action prediction without intermediate structural reasoning makes them brittle under layout changes, especially with heterogeneous virtual keyboards and dynamic content updates.

Our work takes a different approach by decoupling perception and control: a lightweight VLM parses the GUI and a Bayesian navigation module-based on Multinomial–Dirichlet Modeling, handles keyboard-based video search under uncertainty. This modular design supports continual learning, improves cross-app generalization, and meets the computational constraints of on-device Smart TV deployment.

TVAgent Overview

The goal of TVAgent is to autonomously execute user requests on a Smart TV, starting usually from audio input and ending with a completed navigation or search action. Figure 1 provides an overview of the pipeline.

The user issues a voice command (e.g., "Play Interstellar"). This input is processed by an Automatic Speech Recognition (ASR) module to produce text, followed by an intent recognition module that determines the target action. These components are based on prior work in multimodal dialogue and voice-driven agent systems (Radford et al. 2023; Chen, Zhuo, and Wang 2019) and are not the focus of this paper.

The recognized intent is sent to the action module, which works with the Vision–Language Model (VLM) to parse the current TV screen. Parsing starts by determining the screen status—whether the navigation menu is toggled and whether the page is a search interface—followed by extracting detailed structured GUI information such as the menu layout, highlighted elements, and keyboard structure.

The action module then checks if the parsed information is sufficient to generate the action sequence. If so, it outputs the sequence directly; otherwise, it performs a few targeted exploration steps to gather missing context. To reduce redundant exploration—thereby improving both efficiency and user experience—the system stores the navigation menu contents and keyboard adjacency map for each application. This paper focuses on two pipeline components: 1. VLM for GUI parsing, optimized for on-device use across diverse Smart TV applications. 2. Action module, which produces accurate, context-aware action sequences.

Within the TVAgent framework, our method addresses three core navigation functions:

1. **Page-level navigation:** Moving between major interface contexts within an app.
2. **Intra-page content navigation:** Selecting video content within the same page.
3. **Virtual keyboard–based video search:** Typing a user-specified query on an on-screen keyboard.

GUI Parsing

In this section, we detail our approach to GUI parsing for Smart TV environments. While existing GUI datasets (Chen et al. 2024) and GUI-agent frameworks (Qin et al. 2025; Lin et al. 2025) have advanced the field for computer and mobile interfaces, they lack critical TV-specific scenarios such as virtual keyboards and the high density of video thumbnails. These domain discrepancies make it necessary to adapt existing GUI-agent methodologies to meet the unique requirements of Smart TV interaction.

To enable VLM to achieve accurate, robust, and generalizable Smart TV GUI understanding—without relying on extensive manual data collection and annotation—we introduce a synthetic dataset tailored for this domain. Our automated data generation pipeline produces realistic Smart TV screenshots with ground-truth labels for both content information (for GUI parsing) and spatial information (for grounding or segmentation tasks). To further streamline integration with downstream components, all outputs are formatted in *JSON*, allowing the action module to efficiently retrieve and utilize the parsed data.

Automated Generation of Training Data

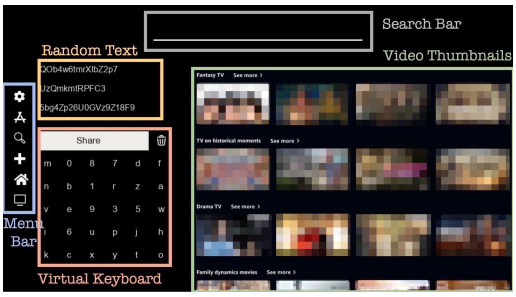


Figure 2: An example of the synthetic data of Smart TV search page screenshot with a virtual keyboard.

Smart TV screens typically consist of distinct components, which we leverage in a two-stage dataset generation process: generating individual components, and assembling them with randomized layouts to produce diverse composite images. This structure enables automatic annotation, as component metadata is tracked during generation. All targets are stored in JSON format for direct integration with downstream modules.

Video thumbnail images. Each page contains three main elements: video row titles, video thumbnails, and a toggleable left menu. We curated 7,488 thumbnails, grouped by aspect ratio into rows, then arranged these rows on a large canvas. Random cropping of TV-sized regions from this canvas simulates page scrolling. Each cropped and uncropped image was parsed with *Qwen2.5-VL*[1] (Bai et al. 2025). One thumbnail per page was randomly highlighted using two distinct visual styles. Row titles and menu text were drawn from a GPT-4o–generated pool (90% probability) or generated randomly (10%).

Search page screenshots. Each search page includes a virtual keyboard, left menu, search bar, operational text/recommendations, and thumbnails (Fig. 2). Although parsing focuses on keyboard layout and selection, including all components prevents learning shortcut and improves robustness. To enhance generalization, keyboard layouts, formats, and positions are randomized. Further details are in the appendix.

Implementation Details

Given the limited hardware resources available on Smart TVs, we selected **LLAVA-OneVision-Qwen2-0.5B** as the base model for GUI parsing due to its compact size and efficient inference. In the data generation, we collected 147 icon images across 14 types and 40 different fonts. (We exclude the fonts that are too thin which lead poor visual performance on the screen). Our training dataset consists of 150,000 video thumbnail images (140,000 without the toggled left menu and 10,000 with the toggled menu) and 100,000 search page images (50,000 images with fully random keyboard layout, and 50,000 images with alphabetical keyboard layout), all rendered at a resolution of 960×540 pixels.

We fine-tuned both the vision encoder and language model components of the base model using LoRA (Hu et al. 2022) with a rank of 8, resulting in a total of 11.7M trainable parameters. Following this large-scale synthetic training stage, we performed few-shot fine-tuning using data from six real-world Smart TV applications to improve domain adaptation. For each of the stage we train the model with 5 epochs. Additional training details are provided in the Appendix.

Action Module

The action module is responsible for generating the sequence of remote-control commands needed to fulfill a recognized intent. It supports three navigation functions: page-level navigation, intra-page content navigation, and virtual keyboard–based video search.

Page-level navigation moves between major interface contexts within an application, such as from the home page to the search page or from a movie page to a live sports page. This task typically begins with a one-time exploration of the left-side navigation menu, after which the agent can directly switch between menu elements.

Intra-page content navigation involves selecting video content within the same page. A fine-tuned VLM parses thumbnail contents and their relative positions into a structured JSON representation. Navigation between videos then

[1]https://github.com/QwenLM/Qwen2.5-VL

reduces to traversing a spatial grid derived from this parsed layout.

Virtual keyboard–based video search remains uncertain even after the keyboard layout (key boxes and labels) has been parsed. The core difficulty is state-dependent transitions: when moving the focus off a multi-span key (e.g., space or delete) onto neighboring single-span keys, the next selection can have multiple valid outcomes. In practice, the destination may depend on the entry point onto the multi-span key (i.e., which edge or subregion was previously selected), so geometry alone does not determine a unique next key. This behavior introduces ambiguity that a single image (and thus a single pass of VLM predictions) cannot resolve, making one-shot action-sequence planning unreliable if we rely only on per-image VLM outputs. To motivate our use of the *adjacency map* (AM) and *coordinate-to-key* (C2K) representations, Figure 3 shows why the parsed layout alone is insufficient. The example captures the state-dependent behavior of multi-span keys by specifying (i) a mapping from visual coordinates to key identities (C2K) and (ii) directional transition constraints between keys (AM). While one could model the keyboard with a higher-order (multi-stage) Markov chain, the AM + C2K factorization provides a straightforward, interpretable initialization from the parsed layout and supports efficient Bayesian updates during interaction.

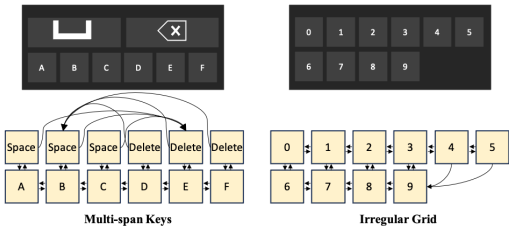


Figure 3: Illustration of irregular key layouts and multi-span keys in Smart TV applications. Arrows indicate non-standard directional transitions, highlighting challenges in modeling consistent navigation paths.

To support one-shot prediction of the action sequence given a keyboard representation and a target query, we explicitly model two structures beyond the layout: an *adjacency map* (AM) that captures valid directional transitions between keys (allowing these transitions to be probabilistic and state-dependent), and a *coordinate-to-key mapping* (C2K) that links visual key regions to semantic key identities. We estimate and refine AM and C2K using a *Multinomial–Dirichlet Modeling* (MDM) framework (Blei, Ng, and Jordan 2003; Harrison et al. 2020). In this Bayesian formulation, VLM-guided exploration supplies categorical observations of key-to-key transitions and key identities; Dirichlet priors yield posteriors that are updated online. This lets the agent disambiguate history-dependent transitions without ground-truth layouts and quickly adapt its AM/C2K to new IME designs, enabling robust, feed-forward (one-shot) planning of the full typing sequence. Preliminaries for MDM are provided in the supplementary material.

MDM-based Keyboard Structure Refinement

In our framework, the VLM provides a noisy initialization of the *adjacency map* (AM) and *coordinate-to-key* (C2K) associations, which we use as priors. We then refine these priors with Multinomial–Dirichlet Modeling (MDM), incorporating interaction data accumulated during use to adapt the model over time.

Each keyboard coordinate i maintains two belief distributions: (i) a categorical distribution over key identities $p(k \mid i)$ (C2K), and (ii) for each navigation action $a \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$, a categorical distribution over next keys $p(j \mid i, a)$ (AM). Both are modeled with multinomial likelihoods and Dirichlet priors parameterized by concentration vectors $\boldsymbol{\alpha}$. The VLM seeds these priors by assigning a confidence weight ρ to the predicted bin (e.g., `a` at coordinate i). As interaction data arrive, we apply weighted count updates to obtain refined posteriors.

Because VLM predictions are single-shot and may miss layout-specific irregularities—such as multi-span keys or state-dependent focus transitions—visual cues alone are insufficient. Interaction traces (e.g., realized cursor moves and selected keys) provide empirical evidence in the form of count vectors, enabling principled Bayesian updates of the Dirichlet posteriors via conjugacy.

We consider three deployment settings for MDM updates: *offline*, *online*, and *hybrid*. All settings share a common path-finding module based on Breadth-First Search (BFS), with the VLM supplying the initial focus location.

Learning Regimes

We consider three learning regimes that differ in how the Multinomial–Dirichlet posteriors for AM/C2K are initialized and updated.

Offline: given a predefined set of training keywords, we compute pre-planned BFS paths on the initial AM/C2K; the traversed identities and transitions act as pseudo-observations to update Dirichlet posteriors, which are then *fixed* at deployment—yielding fast runtime but requiring a pretraining phase and offering limited adaptivity.

Online: with no prior logs, the agent starts from VLM-seeded priors and performs real-time BFS while updating posteriors after each realized move; early performance may be suboptimal but rapidly improves, and BFS remains lightweight on small grids.

Hybrid (default): a brief offline warm-start using a single, challenging keyword reduces initial uncertainty, followed by continued online updates during use—achieving near–offline start-up speed with online adaptability and minimal overhead.

The difference between offline and online regimes is illustrated in Figure 6.

Dynamic Knowledge Base for Platform Adaptation

A key component for deployability is the Dynamic Knowledge Base (DKB), which enables the agent to adapt to unseen applications and reduce redundant parsing. The DKB's function is to cache the learned AM and C2K mappings for each application.

Model	Params	Homepage Parsing			Navigation Menu Parsing		Virtual Keyboard Parsing		
		Row titles	Thumbnail-Content	Selection	Menu-Content	Selection	Key-Layout	Key-Span	Selection
GPT-4o	>100B	0.898	0.518	0.801	0.944	0.736	0.352	0.332	0.900
UI-TARS-1.5	7B	0.691	0.253	0.371	0.238	0.521	0.338	0.304	0.780
TVAgent	0.9B	**0.917**	**0.770**	**0.811**	**0.967**	**0.975**	**1.000**	**0.985**	**0.990**

Table 1: Performance comparison of TVAgent, GPT-4o (Hurst et al. 2024), and UI-Tars (Qin et al. 2025) across eight tasks in three scenarios on four widely used apps. For *homepage parsing*, we report string similarity for row titles and thumbnail content using $1-$ normalized Levenshtein distance (i.e., $s = 1 - \frac{d_{\text{Lev}}(\hat{y},y)}{\max(|\hat{y}|,|y|)}$), and the accuracy of current-selection detection (threshold 0.75). For *navigation menu parsing*, we report exact-match accuracy for both the menu content and the current selection. For *virtual keyboard parsing*, *keyboard layout* is the row-wise matching accuracy (fraction of rows perfectly detected), and *key span* is the accuracy for keys whose content and span are both correct.

Navigation Function	**Success Rate**
Page-level navigation	0.977
Intra-page content navigation	0.830
Virtual keyboard-based video search	1.000

Table 2: Success rates of TVAgent's core navigation functions, evaluated across four real-world applications.

To operate in cases where a direct TV API is unavailable, the DKB can identify platforms by performing online, density-based clustering on mean-pooled visual token embeddings extracted from the keyboard region. If a new screenshot is matched to a known cluster, its cached AM/C2K is loaded; otherwise, a new platform entry is initialized. This allows the agent to generalize to new apps and adapt over time. The full methodology for this screenshot-only fallback, including feature extraction and clustering algorithms, is detailed in the Appendix and Algorithm 2.

Implementation Details

We implement our **Hybrid (default)** learning regime by leveraging the DKB. Upon first access to an unseen app (as identified by the DKB), the VLM parses the on-screen keyboard to provide initial priors for the AM and C2K mapping. We then perform a brief offline warm-start: a single MDM pass using a fixed 30-key sequence. The resulting AM/C2K posteriors are cached in the DKB.

For later searches on an app already in the DKB, we initialize AM/C2K from the cached posteriors and run real-time BFS planning while performing online MDM updates in parallel. Early stopping terminates online updates once confidence is high (e.g., no errors for 7 consecutive moves or posterior entropy below a threshold). When early stopping is triggered, we simply execute BFS on the cached AM/C2K (no further adaptation), producing the action sequence for the target query. This design achieves near-zero warm-start overhead while retaining adaptability to layout drift.

Experiments

GUI Parsing

We evaluated TVAgent on eight tasks across three scenarios using four widely used commercial apps. Our model was compared against GPT-4o (Hurst et al. 2024) and UI-Tars (Qin et al. 2025), both tested with one-shot prompting. The results are shown in Table 1.

For *homepage parsing*, we report string similarity for row titles and thumbnail content using $1-$ normalized Levenshtein distance (i.e., $s = 1-\frac{d_{\text{Lev}}(\hat{y},y)}{\max(|\hat{y}|,|y|)}$), and the accuracy of current-selection detection (threshold 0.75). For *navigation menu parsing*, we report exact-match accuracy for both the menu content and the current selection. For *virtual keyboard parsing*, *keyboard layout* is the row-wise matching accuracy (fraction of rows perfectly detected), and *key span* is the accuracy for keys whose content and span are both correct.

From the results, TVAgent significantly outperforms compared to the baselines in homepage thumbnail-content detection, current selection detection in navigation menus, and both keyboard layout and key-span parsing. On the remaining tasks, TVAgent delivers comparable or superior performance, despite using substantially fewer parameters.

Agent Capability Evaluation

We evaluate **TVAgent** on its three core functions, with results on four real-world apps summarized in Table 2.

Page-level navigation was tested over 1,680 trials, measuring the accuracy of reaching a target page from a random start.

Intra-page content navigation was tested over 4,922 trials, measuring the accuracy of reaching a target movie thumbnail from a random start.

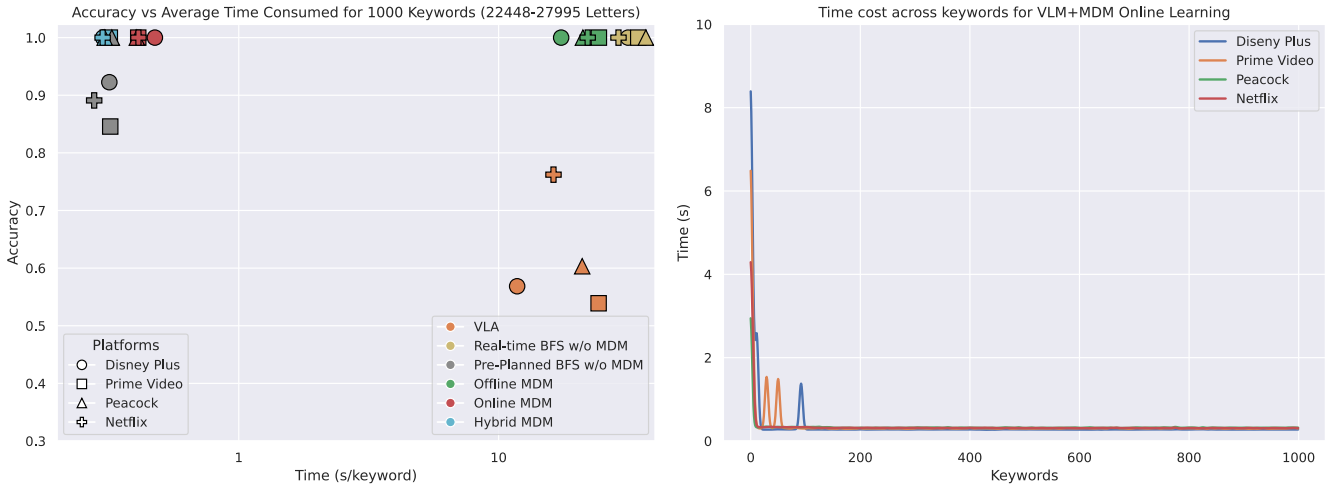
Virtual keyboard–based video search was tested for accuracy over 1,000 randomly sampled movie titles from the MovieLens dataset (Harper and Konstan 2015).

Ablation

Ablation on Synthetic Data. We assess the effectiveness of synthetic data for improving GUI parsing via an ablation on three training regimes, all fine-tuning *LLAVA-OneVision-Qwen2-0.5B* with LoRA: *real-only*, *synthetic-only*, and *mixed (real+synthetic)*. For *real-only* and *synthetic-only*, we train for 5 epochs. For the *mixed* regime, we perform a staged procedure: (i) 5 epochs on synthetic data, then (ii) further fine-tune on real data with a new LoRA initialized from the synthetic-adapted checkpoint (LoRA stacking/merging). Training uses 6 apps; evaluation uses 4 held-out apps.

Method	Role of VLM	Frequency of VLM Prediction	AM/C2K	Training Stage	Test-time training
VLA	Generate Action	Once per letter	-	×	×
Pre-Planned BFS w/o MDM	Predict Current Key	Once per keyword	Initial Guess	×	×
Real-time BFS w/o MDM	Predict Current Key	Every Movement	Initial Guess	×	×
Offline MDM	Predict Current Key	Once per keyword	MDM	✓	×
Online MDM	Predict Current Key	Every Movement (Early Stopping) + Once per keyword	MDM	×	✓
Hybrid MDM	Predict Current Key	Every Movement (Early Stopping) + Once per keyword	MDM	△	✓

Table 3: Comparison of keyboard navigation approaches. VLA serves as a baseline, while ablation studies evaluate the effect of removing MDM in both offline and online settings, denoted as *Pre-Planned* and *Real-time BFS w/o MDM*, respectively. *Offline MDM* requires a full training phase, whereas *Hybrid MDM* uses a minimal setup (e.g., one training sample for one epoch, marked as △) to initialize the model.



(a) Accuracy vs Time Comparisons **(b) Time cost per keyword during Online MDM**

Figure 4: (a) Time and accuracy comparison across navigation approaches. Colors represent different methods, and markers denote results for four distinct Smart TV apps. The x-axis shows the average time required to process 1,000 test samples, while the y-axis indicates the agent's typing accuracy. (b) Action time per keyword for the Online MDM approach. The x-axis represents the cumulative number of test keywords, and the y-axis shows the time taken per keyword. In early stages, Online MDM incurs higher latency due to updates to AM and C2K from each movement. As the model converges, execution time decreases and stabilizes below 0.4 seconds, requiring only initial cursor detection.

Function	Task	Real Data	Syn Data	Syn+Real
Homepage Parsing	Row titles	0.880	0.784	**0.917**
	Thumbnail-content	0.613	0.665	**0.770**
	Selection	0.643	0.750	**0.811**
Navigation Menu Parsing	Menu-content	0.863	0.596	**0.967**
	Selection	0.871	0.879	**0.975**

Table 4: Ablation of GUI parsing performance across three training data regimes: real-only, synthetic-only, and mixed (real+synthetic). Models are identical except for training data. The same metrics as in Table 1 are used.

As summarized in Table 4, the mixed regime yields the best performance across all tasks. For *homepage parsing*, mixed training improves row-title accuracy to 91.7%, boosts thumbnail-content to 77.0%(+25.2% over real-only), and raises selection accuracy to 81.1%. For *navigation menu parsing*, mixed training achieves 96.7% on menu-content (+10.4%) and 97.5% on selection (+10.4%). These results suggest that synthetic data enhances robustness to visual diversity and focus selection, while real data captures menu semantics and app-specific structure; combining both closes the domain gap and consistently delivers the strongest performance.

Ablation on Keyboard Refinement. We ablate the effect of *offline*, *online*, and *hybrid* MDM on keyboard refinement. For **offline MDM**, we generate 200 synthetic training keywords by sampling 10-character alphanumeric strings (e.g., `al25niem7g`) and, when applicable, randomly inserting functional keys such as `[DELETE]` or `[CLEAR]` (e.g., `al 25[DELETE]ni em7g`). AM and C2K are updated at the end of each epoch for 10 epochs using pre-planned BFS trajectories as pseudo-observations. For **online MDM**, there is no pretraining; the agent starts from VLM-seeded priors and learns entirely from real-time interaction. For **hybrid MDM**

(our default), we warm start with a single 30-key keyword trained for one epoch, then switch to online updates during use. Evaluation uses 1,000 movie titles randomly sampled from MovieLens (Harper and Konstan 2015), augmented with random functional-key requests (e.g., `toy story` $\rightarrow$ `toy st[DELETE]or[CLEAR]y`) to mimic realistic edits. Unless otherwise noted, Dirichlet priors are initialized uniformly with concentration weight ρ=2, and the BFS planner uses a movement penalty of -1 per incorrect cursor step. Early stopping halts online updates once confidence is high (e.g., zero navigation errors for τ consecutive moves or low posterior entropy).

Figure 4(a) shows that all three MDM variants (*offline*, *online*, *hybrid*) reach **100% accuracy** on the 1,000-title benchmark. Non-MDM baselines exhibit a speed–accuracy trade-off: real-time BFS without early stopping incurs higher latency, while adding early stopping substantially reduces inference time toward the fastest baseline, but still lags the MDM approaches. Pure VLA action-prediction models perform poorly on this task, with both lower accuracy and higher latency, indicating that direct sequence prediction is suboptimal for keyboard navigation. Figure 4(b) details *online MDM* time per query: initial interactions spend several seconds refining AM/C2K; as posteriors converge, updates cease and per-query navigation stabilizes below **0.4 s**. Overall, MDM-based refinement achieves perfect accuracy while maintaining competitive runtime, and the *hybrid* regime provides the best balance of fast start-up and continued adaptability.

Discussion on Deployment and Usability

To contextualize the practical contribution of TVAgent for real-world systems, we discuss its intended deployment path and its specific, complementary role in the modern Smart TV ecosystem.

Path to Deployment and Generalizability

We envision TVAgent being deployed at the operating system (OS) level of a Smart TV. Unlike solutions that require per-app integration, TVAgent is designed as a universal, vision-based controller.

1. **Standardized Interface:** The system's core pipeline is intentionally modular. The VLM parses the screen's visual-only feed and outputs a structured JSON representation of the GUI. The Action Module then consumes this JSON and produces a sequence of basic, discrete remote-control commands (e.g., UP, DOWN, LEFT, RIGHT).
2. **Cross-Application Generality:** Because this input (pixels) and output (D-pad commands) are universal, the agent can operate across *any* application on the TV, whether it is a streaming service, a settings menu, or a live TV guide. This commitment to "cross-app generalization" is a foundational design choice to handle heterogeneous layouts.
3. **Adaptation to Unseen Platforms:** The system is explicitly designed to handle new, "unseen" applications. Our experiments validate this: the model was trained on data from six applications and evaluated on four entirely **held-out applications**. The Dynamic Knowledge Base (DKB) is the key mechanism for this. Upon encountering an unseen app, the DKB's online clustering module identifies the new platform, parses its unique layout, and caches the learned keyboard structure (Adjacency Map and Coordinate-to-Key mapping) for all future interactions. This allows the agent to adapt and improve its efficiency over time without manual intervention.

Beyond App-Specific Controls

While modern interfaces like voice assistants and "magic pointer" remotes have improved content search, their functionality is often integrated on a per-application basis rather than at the OS level.

- **Voice commands** excel at direct-access tasks (e.g., "Play 'Interstellar"'), but their ability to perform deep, multi-step navigation is typically limited to the specific streaming apps that have explicitly enabled such integration.
- **Pointer remotes** are not a standard feature on all hardware and are also subject to varying levels of support within different applications.

TVAgent is designed to be a **universal agent** that operates at the OS level, specialized for the discrete, directional action space that remains the fundamental interaction method across *all* Smart TV applications. It provides a robust, generalizable solution for fine-grained navigation and text-entry tasks, regardless of the specific app's internal design. The 100% success rate on keyboard-based video search—a noted usability barrier—demonstrates its effectiveness in solving this persistent, universal challenge.

Conclusion

We presented **TVAgent**, a practical vision–language agent for Smart TV control that combines (i) a fine-tuned VLM for TV GUI parsing, (ii) a Bayesian keyboard controller via Multinomial–Dirichlet Modeling (MDM), and (iii) a dynamic knowledge base (DKB) that amortizes exploration across apps. On real apps, TVAgent achieves **97.7%** success in page-level navigation and **100%** accuracy for virtual keyboard search with sub-second latency (stabilizing < 0.4 s/query). Ablations show that *mixed* (real+synthetic) training consistently improves GUI parsing, and that the *hybrid* MDM regime delivers perfect accuracy with fast start-up and low steady-state cost.

This VLM+MDM+DKB architecture demonstrates a clear path toward a deployable, OS-level agent that generalizes to unseen applications, solving navigation challenges unaddressed by app-specific controls. From an applied standpoint, the system is lightweight and data-efficient. The DKB, in particular, ensures robustness to heterogeneous IME layouts by reusing learned structures, reducing user-perceived latency without manual rules.

Future work includes expanding multilingual coverage, integrating on-device ASR, and extending continual learning for layout drift and personalization.

References

Alam, I.; Khusro, S.; and Khan, M. 2019. Usability barriers in smart TV user interfaces: A review and recommendations. In *2019 international conference on Frontiers of Information Technology (FIT)*, 334–3344. IEEE.

Baechler, G.; Sunkara, S.; Wang, M.; Zubach, F.; Mansoor, H.; Etter, V.; Cărbune, V.; Lin, J.; Chen, J.; and Sharma, A. 2024. Screenai: A vision-language model for ui and infographics understanding. *arXiv preprint arXiv:2402.04615*.

Bai, S.; Chen, K.; Liu, X.; Wang, J.; Ge, W.; Song, S.; Dang, K.; Wang, P.; Wang, S.; Tang, J.; et al. 2025. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*.

Blei, D.; Ng, A.; and Jordan, M. 2003. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3.

Chen, D.; Huang, Y.; Wu, S.; Tang, J.; Chen, L.; Bai, Y.; He, Z.; Wang, C.; Zhou, H.; Li, Y.; et al. 2024. Gui-world: A video benchmark and dataset for multimodal gui-oriented understanding. *arXiv preprint arXiv:2406.10819*.

Chen, Q.; Zhuo, Z.; and Wang, W. 2019. BERT for Joint Intent Classification and Slot Filling. *arXiv preprint arXiv:1902.10909*.

Chen, W.; Cui, J.; Hu, J.; Qin, Y.; Fang, J.; Zhao, Y.; Wang, C.; Liu, J.; Chen, G.; Huo, Y.; et al. 2025. GUICourse: From General Vision Language Model to Versatile GUI Agent. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 21936–21959.

Cui, C.; Ma, Y.; Cao, X.; Ye, W.; Zhou, Y.; Liang, K.; Chen, J.; Lu, J.; Yang, Z.; Liao, K.-D.; et al. 2024. A survey on multimodal large language models for autonomous driving. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, 958–979.

Harper, F. M.; and Konstan, J. A. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4): 1–19.

Harrison, J. G.; Calder, W. J.; Shastry, V.; and Buerkle, C. A. 2020. Dirichlet-multinomial modelling outperforms alternatives for analysis of microbiome and other ecological count data. *Molecular ecology resources*, 20(2): 481–497.

Hong, W.; Wang, W.; Lv, Q.; Xu, J.; Yu, W.; Ji, J.; Wang, Y.; Wang, Z.; Dong, Y.; Ding, M.; et al. 2024. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 14281–14290.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; Chen, W.; et al. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2): 3.

Hurst, A.; Lerer, A.; Goucher, A. P.; Perelman, A.; Ramesh, A.; Clark, A.; Ostrow, A.; Welihinda, A.; Hayes, A.; Radford, A.; et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Khan, M.; Khusro, S.; Alam, I.; Ali, S.; and Khan, I. 2022. Perspectives on the design, challenges, and evaluation of smart TV user interfaces. *Scientific Programming*, 2022(1): 2775959.

Li, K.; Meng, Z.; Lin, H.; Luo, Z.; Tian, Y.; Ma, J.; Huang, Z.; and Chua, T.-S. 2025. Screenspot-pro: Gui grounding for professional high-resolution computer use. *arXiv preprint arXiv:2504.07981*.

Li, T. J.-J.; Popowski, L.; Mitchell, T.; and Myers, B. A. 2021. Screen2vec: Semantic embedding of gui screens and gui components. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 1–15.

Lin, K. Q.; Li, L.; Gao, D.; Yang, Z.; Wu, S.; Bai, Z.; Lei, S. W.; Wang, L.; and Shou, M. Z. 2025. Showui: One vision-language-action model for gui visual agent. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, 19498–19508.

Liu, G.; Zhao, P.; Liu, L.; Chen, Z.; Chai, Y.; Ren, S.; Wang, H.; He, S.; and Meng, W. 2025. Learnact: Few-shot mobile gui agent with a unified demonstration benchmark. *arXiv preprint arXiv:2504.13805*.

Maniyar, S.; Trivedi, V.; Mondal, A.; Mishra, A.; and Jawahar, C. 2025. AI-Generated Lecture Slides for Improving Slide Element Detection and Retrieval. *arXiv preprint arXiv:2506.23605*.

Peng, Y.-H.; Huq, F.; Jiang, Y.; Wu, J.; Li, X. Y.; Bigham, J. P.; and Pavel, A. 2024. Dreamstruct: Understanding slides and user interfaces via synthetic data generation. In *European Conference on Computer Vision*, 466–485. Springer.

Qin, Y.; Ye, Y.; Fang, J.; Wang, H.; Liang, S.; Tian, S.; Zhang, J.; Li, J.; Li, Y.; Huang, S.; et al. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*.

Radford, A.; Kim, J. W.; Xu, T.; Brockman, G.; McLeavey, C.; and Sutskever, I. 2023. Robust Speech Recognition via Large-Scale Weak Supervision. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, 28492–28518. PMLR.

Rahman, A.; Chawla, R.; Kumar, M.; Datta, A.; Jha, A.; NS, M.; and Bhola, I. 2024. V-zen: Efficient gui understanding and precise grounding with a novel multimodal llm. *arXiv preprint arXiv:2405.15341*.

Sapkota, R.; Cao, Y.; Roumeliotis, K. I.; and Karkee, M. 2025. Vision-language-action models: Concepts, progress, applications and challenges. *arXiv preprint arXiv:2505.04769*.

Wang, Z.; Chen, W.; Yang, L.; Zhou, S.; Zhao, S.; Zhan, H.; Jin, J.; Li, L.; Shao, Z.; and Bu, J. 2025. Mp-gui: Modality perception with mllms for gui understanding. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, 29711–29721.

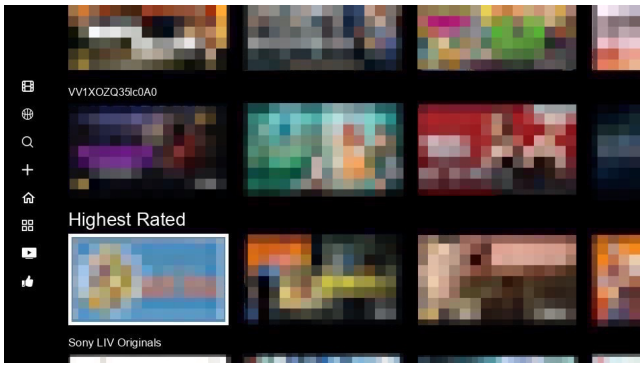
Yang, B.; and Li, S. 2023. GUI2DSVec: Detecting Visual Design Smells Based on Semantic Embedding of GUI Images and Components. In *International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, 445–457. Springer.

Zhang, B.; Shang, Z.; Gao, Z.; Zhang, W.; Xie, R.; Ma, X.; Yuan, T.; Wu, X.; Zhu, S.-C.; and Li, Q. 2025. TongUI: Building Generalized GUI Agents by Learning from Multimodal Web Tutorials. *arXiv preprint arXiv:2504.12679*.

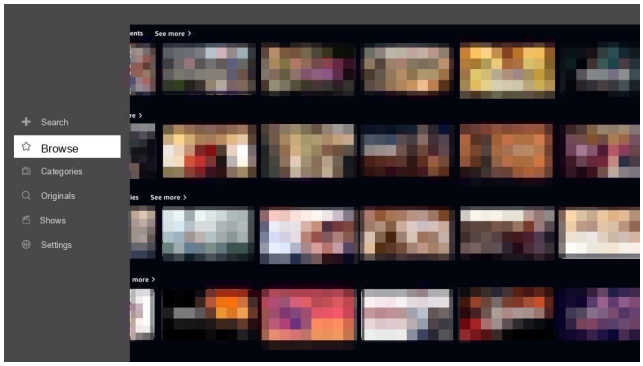
Expanded Methods Description

TV GUI Parsing

We first provide the parsing prompts and expected outputs for each screen type in Table 5. Due to commercial licensing constraints, we do not include real-data examples; instead, we showcase additional synthetic examples in Figure 5.



(a) Synthetic homepage screenshot



(b) Synthetic expanded menu screenshot

(c) Synthetic search page screenshot

Figure 5: More examples of synthetic data.

Beyond converting visual content into structured *JSON*, we also equip the action module with screen-type awareness so it can select the appropriate VLM prompt. To this end, we fine-tune the VLM to classify the current screen via a multiple-choice prompt with three options: (i) extended menu, (ii) virtual keyboard, or (iii) neither. This coarse screen classification is more efficient than detecting each UI element individually. When both a menu and a keyboard are present, we prioritize the extended menu as the ground truth. To support navigation from a menu to the search screen, we then issue a single consolidated prompt that simultaneously extracts the menu layout and the currently selected item. The resulting two-stage video-search pipeline is shown in Figure 7.

MDM-based Keyboard Structure Refinement

Preliminaries for Multinomial Dirichlet Modeling MDM is grounded in Bayesian inference where we infer the posterior distribution of categorical variables using count-based updates. Specifically, we model the true key label at each coordinate as a categorical random variable $K_{ij} \in \Sigma$, and the directional neighbor of a given key as $N_{ij,\text{dir}} \in \mathcal{C}$, where Σ is the key vocabulary and $\mathcal{C}$ is the set of possible coordinate transitions (e.g., up, down, left, right).

Let $\theta = (\theta_1, \ldots, \theta_k)$ denote the unknown categorical distribution over k categories. We place a Dirichlet prior over θ with hyperparameters $\alpha = (\alpha_1, \ldots, \alpha_k)$:

$$p(\theta) = \text{Dir}(\theta \mid \alpha) = \frac{1}{\text{Beta}(\alpha)} \prod_{i=1}^{k} \theta_i^{\alpha_i - 1},$$

where

$$\text{Beta}(\alpha) = \frac{\prod_{i=1}^{k} \Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^{k} \alpha_i\right)}.$$

Given n observations of categorical outcomes resulting in count vector $\mathbf{x} = (x_1, \ldots, x_k)$, the likelihood is modeled by the multinomial distribution:

$$p(\mathbf{x} \mid \theta) = \frac{n!}{x_1! \cdots x_k!} \prod_{i=1}^{k} \theta_i^{x_i}.$$

Applying Bayes' rule, the posterior is:

$$p(\theta \mid \mathbf{x}) \propto p(\mathbf{x} \mid \theta) \cdot p(\theta) \propto \prod_{i=1}^{k} \theta_i^{x_i + \alpha_i - 1},$$

which is equivalent to a Dirichlet distribution:

$$p(\theta \mid \mathbf{x}) = \text{Dir}(\theta \mid \alpha + \mathbf{x}).$$

This result allows for simple additive updates to the Dirichlet prior based on observed key labels and transitions.

MDM Learning Regimes An overview of the learning regimes is illustrated in Figure 6 and Algorithm 1.

Online MDM. First, we introduce pseudo-code for Online MDM for Keyboard Navigation. The initial Coordination-to-Key mapping (for key) and Adjacency Map (for edge) are obtained by VLM. We initialize prior distributions for key and edge with prior weight ρ. The posterior distribution is updated through a Dirichlet distribution with the observation with parameter α by counting the observation during the navigation.

Task	Query Prompt	Answer Example
Homepage Parsing	You are an expert UI layout parser. Your task is to output a JSON representation of the image. First read top to bottom the row titles. Next read left to right only the movie/TV content names row wise. Then read current_selecti -on. Be concise. Important guidelines:1. Ensure all output is in valid JSON format. 2. Include null values for any text or identifiers that are unclear or unreadable in the image.3. The current_selection should always be populated, identifying which UI element is currently selected or highlighted.Analyze the image thoroughly and provide a compreh -ensive mapping of the OTT app's UI elements, ensuring all visibl -e and inferrable components are represented in the JSON structure	{"row_titles": ["UPCOMING", "LEAGUES", "null"], "content_titles": [["Track and Field Women's Day 1", "2025 PFL: WW & FW", "Mirandes vs. Racing", "Liberty vs. Fev er"], ["NCAA Track & Field", "Professional Fighters Leagu - e", "NCAA Baseball", "UFC", "Formula One", "Northwoo -d League Baseball", "The New England Collegiate Baseball League"], ["null"]], "current_selection": "Track and Field Wo -men's Day 1"}
Navigation Menu Parsing	Parse the left menu page. Read all the elements in the menu on the left. List the names of the elements of the menu in vertical order (to -p to bottom) in JSON format. Include left_menu with all the elem -ents in the left menu ordered top to bottom, current_section to hig -hlight what's the section in the background, current_selection indi -cating what's the currently selected option on the left_menu. Be concise.	{"left_menu": ["Joseph SWITCH", "SEARCH", "ORIGINA LS", "WATCHLIST", "MOVIES", "SETTINGS", "SERIES" , "HOME"], "current_selection": "Joseph SWITCH"}
Virtual Keyboard Parsing	Parse the search screen. List the keys from the keyboard in JSON format, info about the span of larger keys, current_selection. Format the li -st of keys with one list per row.	{'layout': [['A','B','C','D' 'E','F'], ['G','H','I','J' 'K','L'], ['M','N','O','P' 'Q','R'], ['S','T','U','V' 'W','X'], ['Y','Z','0','1' '2','3'], ['4','5','6','7' '8','9'], ['Space','Delete']], { 'info': {'span': {'Space':3, 'Delete':3 }}}}
Analyze Screen Status	Analyze the screenshot and respond based on the following condi -tions: If the left menu bar is expanded, respond with A. If not, an -d a virtual keyboard is present, respond with B. Otherwise, respo -nd with C. Return answer in [A/B/C]. A: Expanded Menu, B: Ke -yboard, C: Others.	B

Table 5: Input prompts and answer format for training dataset.

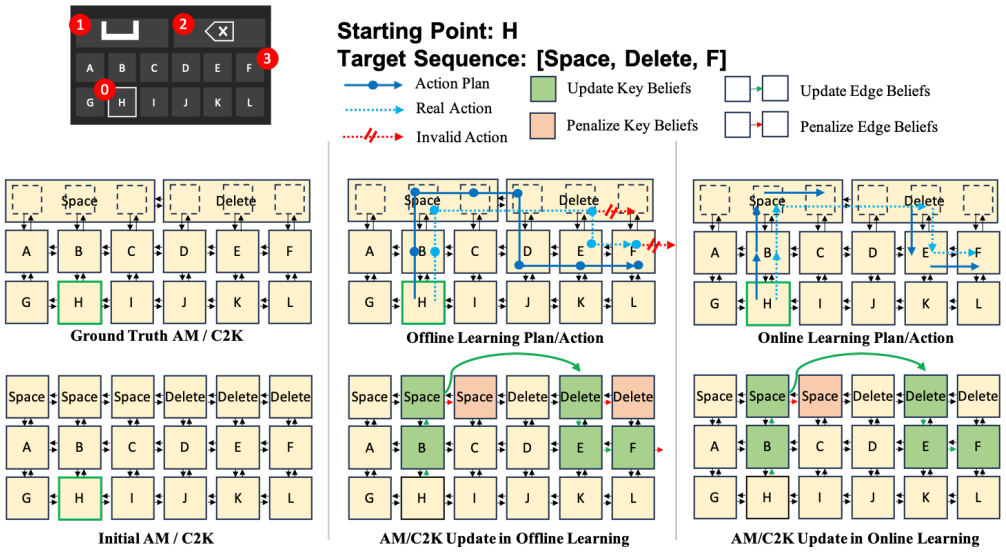


Figure 6: Comparison of Offline (Pre-planned) and Online (Real-time) Learning Processes. In the offline setting, AM and C2K are updated using training keywords, with a single BFS path planned per character. In the online setting, BFS is executed iteratively during inference, allowing AM and C2K to be dynamically refined through real-time interaction. The upper panel illustrates the planned action sequence based on estimated structures versus the executed actions based on ground truth. The lower panel depicts how belief distributions are updated through comparisons between predicted and observed transitions.

Algorithm 1: Online Multinomial-Dirichlet Modeling for Keyboard Navigation

Input: (CK, AM): Initial guess of coordination-to-key mapping and adjacency mapping, ρ: weighting parameter, $\mathcal{T}$: a set of typing keyword, F: Vision-Language Model
Output: Refined (CK, AM), Typed $\mathcal{T}$

Initialize key prior $\theta_{ij}^{key} \sim Dir(\alpha_{ij})$ where $\alpha_{ij}[CK_{ij}] = \rho, \alpha_{ij}[l] = 0 \quad \forall l \neq CK_{ij}$
Initialize edge prior $\theta_{ij,dir}^{edge} \sim Dir(\alpha_{ij,dir})$ where $\alpha_{ij,dir}[AM_{ij,dir}] = \rho, \alpha_{ij,dir}[c] = 0 \quad \forall c \notin AM_{ij}$
for $i, \ldots, E$ **do**
  $\hat{CK}, \hat{AM} \leftarrow \mathsf{MAP}(\theta^{key}, \theta^{edge})$ // Maximum a Posteriori estimate for current state
  for $t \in \mathcal{T}$ **do**
    // For each keyword t
    Initialize Random Cursor $coord\_init$
    $coord \leftarrow \hat{CK}(F(coord\_init))$
    for $\{k \in 0, \ldots, |t| - 1\}$ **do**
      // For each letter
      while $\hat{l}_{k+1} = l_{k+1}$ **do**
        $plan \leftarrow \mathsf{BFS}(\hat{CK}(l_k), \hat{CK}(l_{k+1}))$
        if $plan$ doesn't exist **then**
          $\alpha_{ij}[l_{k+1}] += \rho$ for random i, j. // To increase α for missing letter to random location.
        else
          $(coord, act) \leftarrow plan[0]$ // Execute the first action plan
          $new\_coord \leftarrow \mathsf{MOVE}(coord, act)$
          $\hat{l}_{k+1} \leftarrow F(new\_coord)$ // Observation
          $\alpha_{new\_coord}[\hat{l}_{k+1}] += \rho$ // Update the belief of new coordinates having the observed letter.
          $\alpha_{coord,dir}[new\_coord] += \rho$ // Update the belief of edge between older coordinates to the new coordinates.
          $coord \leftarrow new\_coord$
        end if
      end while
    end for
  end for
end for

Additional Method Variants

In the main system, TVAgent obtains the app name directly from the Smart TV API. Here we describe a *screenshot-only* fallback for cases where the API is unavailable or restricted, and how we use it to avoid re-parsing the keyboard layout on every screen.

Motivation. Extracting a full keyboard layout each time adds latency. Instead, we cache per-app platform information—the adjacency map (AM) and coordinate-to-key (C2K)—in a dynamic knowledge base. Given a new screenshot, we first decide whether it belongs to a *known* platform; if so, we reuse the cached AM/C2K and proceed. If not, we identify (or create) the platform via keyboard features and clustering, then cache the learned AM/C2K for subsequent use.

Algorithm 2: Platform Detection

Input: F: VLM model, $x_i \in X$: screenshot images, $N_{i,t}$:the number of image tokens for each sample i, τ_N: image stacking warm up threshold, ϵ: distance threshold to identify a cluster, β: cluster centroid updating parameter **Output**: c_i: Cluster label for each screenshot,
(R_k, L_k) : Stored coordination-to-key mapping and adjacency mapping of each cluster C_k.

$\mathcal{H} \leftarrow [\,]$ // Initialize a bag for hidden states
while True **do**
  $h_i \leftarrow F(x_i)[: N_{i,t}]$ // Extract the final image tokens.
  if $i \leq \tau_N$ **then**
    $\mathcal{H} \leftarrow [\mathcal{H}; h_i]$ // Stack the hidden states to the bag
  else
    if $\mathcal{M}$ does not exist **then**
      $\mathcal{M} \leftarrow \mathsf{O\text{-}DBSCAN}(\mathcal{H}, \epsilon)$ // Run Online DBSCAN
    end if
    $j' \leftarrow \arg\min_{j \in \mathcal{M}} ||h_i - \mathcal{M}_j||_2$ // Find closest old centroid
    if $||h_i - \mathcal{M}_{j'}||_2 < \epsilon$ **then**
      $c_i \leftarrow j'$ // Assign the cluster label to the current image
      $\mathcal{M}_{j'} \leftarrow \frac{\beta \mathcal{M}_{j'} + h_i}{\beta + 1}$ // Update the centroid
    else
      $\mathcal{H} \leftarrow [\mathcal{H}; h_i]$ // Stack the hidden states to the bag
      Repeat line 6-11
    end if
    ******Cluster Assignment Ends******
    if L_k does not exist **then**
      $R_k, L_k = F(x_i, Q_L)$ // Generate a initial guess of the layout, where Q_L is the question prompt for layout extraction
      $R_k, L_k \leftarrow \mathsf{DM}(R_k, L_k)$ // Refine R_k and L_k with Dirichlet-multinomial mapping
    end if
  end if
end while

Keyboard features. From a screenshot, the VLM performs keyboard grounding to obtain the keyboard bounding box. We then locate the VLM's visual tokens that fall inside this box and compute a mean-pooled embedding as the keyboard feature:

$$\mathbf{f} = \frac{1}{|\mathcal{T}_{\text{kb}}|} \sum_{t \in \mathcal{T}_{\text{kb}}} \mathbf{e}_t,$$

where $\mathbf{e}_t$ is the embedding of token t and $\mathcal{T}_{\text{kb}}$ is the set of tokens within the keyboard box. In practice, among ∼4,501 visual tokens per screen, ∼200–300 typically fall inside the keyboard; mean pooling yields a dense, stable representation for clustering.

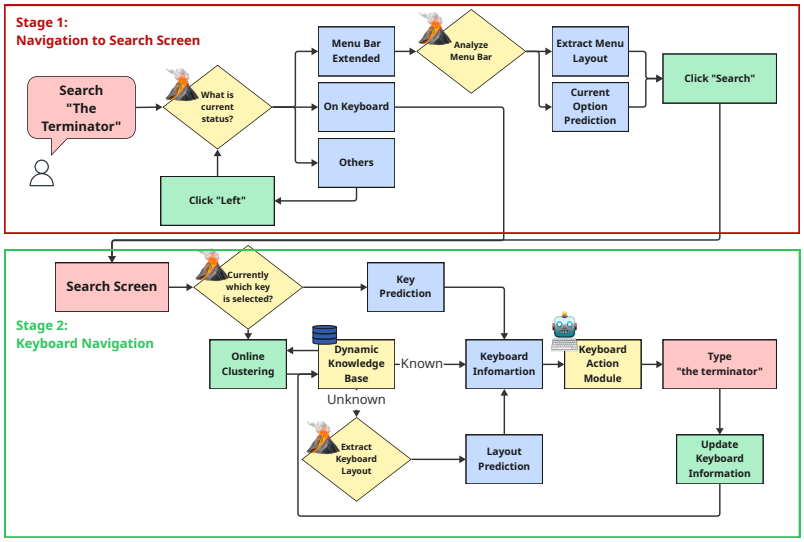


Figure 7: Two-stage GUI agent for Video Searching.

Algorithm 3: Online DBSCAN

Input: $\mathcal{H}$: Stacked Hidden States, ϵ: distance threshold to identify a cluster, $\mathcal{D}$: (Optional) Keyboard Layout Dictionary, $\mathcal{M}$: (Optional) Centroid Dictionary

Output: Cluster assignment, Centroid Dictionary

$\{c_1, c_2, ..., c_N\} \leftarrow$ DBSCAN$(\mathcal{H}, \epsilon)$ where $c_i = k$ denotes instance i is assigned to cluster k
$\mathcal{K} \leftarrow \{k : \exists i, c_i = k\}$ // Get unique cluster labels
$K_{\max} \leftarrow \max(\mathcal{K})$
$p \leftarrow K_{\max} + 1$ // Initialize pseudo-label counter
for $k \in \mathcal{K}$ **do**
  $\mu_k \leftarrow \frac{1}{|\{i:c_i=k\}|} \sum_{i:c_i=k} h_i$ // Calculate centroid μ_k of cluster k
  if $k \neq -1$ **then**
    if $\mathcal{M}$ exists **then**
      $j' \leftarrow \arg\min_{j \in \mathcal{M}} ||\mu_k - \mathcal{M}_j||_2$ // Find closest old centroid
      if $||\mu_k - \mu_{j'}||_2 < \epsilon$ **then**
        $\mathcal{M}_j \leftarrow \mu_k$ // Update the older centroid
      else
        $\mathcal{M}_p \leftarrow \mu_k, p \leftarrow p + 1$ // Assign new cluster's centroid
      end if
    else
      $\mathcal{M}_k \leftarrow \mu_k$ // Assign initial cluster's centroid
    end if
  else
    for each noise point i **do**
      $\mathcal{M}_p \leftarrow \mu_k, p \leftarrow p + 1$ // Assign new cluster's centroid
    end for
  end if
end for

Clustering for platform identity. Because the number of apps (platforms) is unknown and screenshots arrive over time, batch methods that require a fixed cluster count are unsuitable. We therefore adopt a density-based approach (DBSCAN). Classic DBSCAN assumes a static dataset, so we implement an *online* variant: we maintain per-platform prototypes (centroids) and a small buffer of representative features. For an incoming feature $\mathbf{f}$, we assign it to the nearest known prototype if the distance is below a radius ε (and local density criteria are met); otherwise, we initialize a new cluster (new app). Prototypes are updated incrementally (e.g., exponential moving average) as new members arrive. The resulting cluster label serves as the platform ID; when available, a grounded high-confidence app name can be used to name the cluster.

End-to-end use. If a screenshot is matched to a known platform, we load the cached AM/C2K and run BFS without re-extracting layout. If it is deemed new, we initialize a new platform entry, run the keyboard parsing once to seed AM/C2K, and cache the result. This screenshot-only pipeline is lightweight, requires no app instrumentation, and reduces redundant parsing while remaining robust to layout drift.