Tokenized Action Chunks for Autoregressive GUI Drag

Anonymous ACL submission

Abstract

GUI agents must produce drag, lasso, and freehand-stroke trajectories, but most vision-language models emit a single coordinate per forward pass. ShowUI-π ports the π_0 flow-matching action expert from robotics to GUI drag. GUI drag is band-limited: 98% of per-chunk DCT energy on ScreenDrag lies in the first two coefficients, unlike the high-frequency robotic actions that iterative-denoising heads were designed for. A closed-form-decodable tokenizer adapted from FAST fits this regime: its per-coefficient reconstruction bound falls below pixel-scale task tolerance, and the VLM's autoregressive head emits drag trajectories directly under a next-token cross-entropy loss. Under the deployable free-running protocol our 500M-parameter policy beats ShowUI-π at every replanning interval r we tested (horizon $H{=}21$). At the matched-cost setting $r{=}1$ (21 forward passes per drag), Ours reaches 3.82% step-tracking versus ShowUI-π's 1.15%. ShowUI-π stays below 1.5% across all r on its own predictions; its reported 20.12% at $r{=}1$ requires privileged per-step ground-truth cursor injection.

1 Introduction

Drag, lasso, and freehand-stroke gestures are common GUI inputs, but most vision-language models emit a single coordinate per forward pass and discard the intermediate motion that often determines whether a target widget activates. ShowUI-π (Hu et al., 2025) ports the π_0 (Black et al., 2025) recipe from robotics to GUI control: a flow-matching (Lipman et al., 2023) action expert on top of a SmolVLM-2 (Marafioti et al., 2025; Shukor et al., 2025) backbone emits an H-step trajectory in k denoising steps, re-planned at every timestep with only the first action executed.

GUI drag trajectories are strongly band-limited: on ScreenDrag, 98% of per-chunk DCT energy lies in the first two coefficients (§A.2), unlike the high-frequency robotic actions that motivated ShowUI-π's flow-matching expert. The spectral concentration suggests the expert head may be inherited from robotics rather than required for GUI control, especially since the reported 20.12% success at $r{=}1$ relies on per-step ground-truth cursor injection, while free-running execution remains below 1.5% across all r. We therefore replace the action expert with a FAST-style closed-form tokenizer (Pertsch et al., 2025): quantile-normalized DCT coefficients (Ahmed et al., 1974) are discretized, BPE-compressed (Sennrich et al., 2016), and emitted directly by the VLM's autoregressive head, with analytically bounded sub-pixel reconstruction error (§3). The model trains under a single next-token cross-entropy loss across drag, click, and grounding.

On the 505-episode ScreenDrag test set, under the deployable free-running protocol on $H{=}21$, our 500M-parameter policy beats ShowUI-π at every replanning interval r we tested (Fig. 1); at the matched-cost setting $r{=}1$ (21 forward passes per drag), Ours reaches 3.82% step-tracking versus ShowUI-π's 1.15%.

Contributions.

1. We show GUI drag is band-limited (98% energy in the first two DCT coefficients) and fit a FAST-style (Pertsch et al., 2025) closed-form-decodable tokenizer with analytic per-coefficient error bound (§3, §A.2).
2. We train the drag, click, and grounding mixture jointly under a single next-token cross-entropy loss; the executable H-step trajectory decodes in closed form from the emitted token stream (§3).
3. On ScreenDrag (§4) under free-running on $H{=}21$, Ours beats ShowUI-π at every r (Fig. 1, Table 2); ShowUI-π's 20.12% at $r{=}1$ requires privileged per-step cursor injection.

2 Related Work

Action representation for VLA models. Vision-language-action (VLA) models attach an action head to a pretrained VLM backbone. *Continuous* heads pair the VLM with an iterative-denoising expert: π_0 (Black et al., 2025) introduced the flow-matching (Lipman et al., 2023) recipe and SmolVLA (Shukor et al., 2025) ports it to the smaller SmolVLM-2 (Marafioti et al., 2025) backbone. *Discrete-token* heads bin each action dimension and emit actions through the VLM's standard autoregressive head: RT-2 (Brohan et al., 2023) and OpenVLA (Kim et al., 2024) use 256 bins per dimension that overwrite least-used vocabulary tokens, and Magma (Yang et al., 2025) uses the same recipe for GUI navigation. FAST (Pertsch et al., 2025) replaces per-dimension bins with a short correlated token sequence (DCT (Ahmed et al., 1974) + BPE (Sennrich et al., 2016); details in §3.2).

GUI agents and visual grounding. Visual grounding for GUI agents predicts a single coordinate or bounding box per query. SeeClick (Cheng et al., 2024), UGround (Gou et al., 2024), CogAgent (Hong et al., 2024), Ferret-UI (You et al., 2024), ShowUI (Lin et al., 2025), and GUICourse (Chen et al., 2025) all train VLMs to emit one coordinate per forward pass, so a drag collapses to its two endpoints. We reuse the grounding-data mixture but extend the output to multi-step trajectories.

Drag and trajectory prediction in GUI control. To our knowledge, ShowUI-π (Hu et al., 2025) is the only published GUI drag-benchmark policy whose output is a multi-step trajectory rather than two endpoints. We share its task formulation, backbone, and grounding mixture, with a discrete-tokenizer + autoregressive-head pipeline (§3).

3 Method

Our policy is a single autoregressive vision-language model that emits drag and click trajectories the same way it emits text: as discrete token streams. A closed-form-decodable tokenizer adapted from FAST (Pertsch et al., 2025) compresses each trajectory into a short token sequence; drag, click, and grounding share one vocabulary, all emitted through the language-model head under a next-token cross-entropy loss.

3.1 Trajectory Representation

We encode every GUI interaction as a fixed-horizon trajectory in normalized screen coordinates:

$$A = (\mathbf{a}_1, \ldots, \mathbf{a}_H), \quad \mathbf{a}_t = (x_t, y_t, m_t), \qquad (1)$$

where $(x_t, y_t) \in [0,1]^2$ are cursor coordinates and $m_t \in \{\texttt{down}, \texttt{up}\}$ is the mouse-button state. A drag is a run of down steps closed by a single up; a click is a two-step drag padded with up idle. We use this shared tensor shape, as ShowUI-π (Hu et al., 2025) does, so a single head handles both interaction types without task-specific dispatch.

3.2 Action Tokenization

We compress A into a short discrete token sequence with a four-stage pipeline adapted from FAST (Pertsch et al., 2025), applied per chunk: (i) quantile-normalize each channel to $[-1, 1]$ using the 1st and 99th percentiles of the training set; (ii) apply a Type-II discrete cosine transform (Ahmed et al., 1974) per channel along time to obtain a coefficient matrix $C \in \mathbb{R}^{3\times H}$; (iii) scale and round, $\bar{C}_{ij} = \mathrm{round}(\gamma\, C_{ij})$, which zeros out most entries; (iv) flatten the matrix low-frequency-first in column-major order so the model predicts the coarse shape of the trajectory before its fine detail, then compress it with a byte-pair encoder (Sennrich et al., 2016) trained once on the action corpus. The BPE step absorbs the long runs of zeros from step (iii) and produces a token stream $[T_1, \ldots, T_n]$ with $n \ll 3H$ on real drag data. Stages (i), (ii), (iv) are invertible in closed form. Stage (iii) is many-to-one with per-coefficient error bounded by $1/(2\gamma)$, so decoding is closed-form and needs no learned head. Hyperparameters and fitting details are in Appendix A.1.

3.3 Autoregressive Policy

We add the action tokens $\{T_k\}$ to the VLM's vocabulary as V new special symbols $\{[\texttt{ACT}_0], \ldots, [\texttt{ACT}_{V-1}]\}$, with V equal to the BPE vocabulary size (Pertsch et al., 2025). Our backbone is SmolVLM-2-500M (Marafioti et al., 2025). Let $c_t = (o_t, \mathcal{Q}, \mathbf{a}_{t-1})$ denote the context, with o_t the screenshot, $\mathcal{Q}$ the instruction, and $\mathbf{a}_{t-1}$ the previous executed action. The policy then factorizes as

$$\pi_\theta(T_{1:n} \mid c_t) = \prod_{k=1}^{n} p_\theta(T_k \mid T_{<k}, c_t), \qquad (2)$$

and we train it with the standard next-token cross-entropy loss.

Method	Step-track ↑ oracle	Step-track ↑ free	Any@20 ↑ free	ATE ↓ (px)
SmolVLM-2 lang. baseline (zero-shot)$^\sharp$	0.03	0.03	0.59	820.88
GPT-4o (zero-shot)$^\sharp$	0.47	0.47	2.18	280.97
Claude Sonnet 4.5 (zero-shot)$^\sharp$	26.52	26.52	39.60	202.27
ShowUI-π, r=21	0.63	0.13	10.87	302.30
ShowUI-π, r=1‡	20.12	1.15	12.47	70.72
Coord-bin (per-axis quant.), B=256, H=7×3	21.71	12.08	12.08	175.56
Raw-coord-BPE (no DCT), H=7×3	20.10	9.30	13.27	152.28
Ours, H=21 1-decode	11.91	11.72	21.98	291.03
Ours, H=7×3	**27.60**	**19.76**	**30.30**	130.36

Table 1: **ScreenDrag results** (505 episodes). Step-tracking, any-endpoint, ATE. $^\sharp$Single-call zero-shot: first-frame screenshot only, emits one `(start,end)` drag linearly interpolated to the episode step count; oracle=free for these rows (single chunk, no state-refresh boundary). ‡ShowUI-π r=1 oracle uses privileged per-step GT cursor injection; Ours H=7×3 oracle is the coarser analog, refreshing GT only at chunk boundaries (every 7 steps). $^\flat$Read off Fig. 1. Coord-bin and Raw-coord-BPE are matched-pipeline ablations (§3.2, §A). ShowUI-π harness in App. B. ATE omitted for closed-source baselines whose harness emits only endpoints.

Inference. At test time we decode greedily until an end-of-action marker, then invert BPE and DCT in closed form to recover the executed H-step chunk. Because $n \ll 3H$ on real drag data, one decode of n tokens replaces k denoising calls through a separate expert. We report drag latency in §4.

4 Experiments

We ask three questions. (i) Can a single autoregressive VLM beat expert-based policies on drag-heavy UI tasks? (ii) Does action chunking improve the quality–latency trade-off across replanning budgets? (iii) Which parts of the tokenizer matter, and which can be removed?

4.1 Experimental Setup

We evaluate on ScreenDrag (Hu et al., 2025), the GUI drag-and-click benchmark released with ShowUI-π: each example is a screenshot, an instruction, and a target trajectory, and the policy must produce executable mouse actions in normalized screen coordinates. The test set covers five domains (`captcha`, `premiere`, `desktop`, `powerpoint`, `handwriting`) at 101 episodes each (505 total), with up to 21 steps per episode (`premiere` mean 20.4, `handwriting` 18.5).

Metrics. We report **step-tracking accuracy** (per-step within ϵ=20 px of GT; strict per-step reading of ShowUI-π's TEA), **any-endpoint accuracy** (any chunk step within ϵ of the target endpoint; the permissive reading that reproduces ShowUI-π's 78.55% headline), **ATE** (mean Euclidean deviation, in px, after temporal alignment), and **actions per episode (ACT/ep)** (a hardware-agnostic latency proxy). Our primary metric is step-tracking, because ScreenDrag's lasso, slider, and stroke-path tasks reward fidelity along the path, not only at the endpoint.

Replanning protocol. At `replan_interval`=r, the policy decodes a length-H chunk, executes its first r actions, and re-decodes; r=H commits to the chunk. Each decode consumes one previous-state input, so chunk boundaries are the only points where the choice of that input matters. *Oracle previous-state* feeds the ground-truth previous action at each boundary, so error does not compound; this is the protocol the original ShowUI-π paper specifies, and at r=1 reduces to per-step GT cursor injection. *Free-running* starts from a t=0 sentinel and thereafter feeds back the policy's own predicted previous action, so error compounds across the 21-step trajectory; this is the deployable protocol our headline comparisons run under. At r=H the only boundary is t=0, both modes use the sentinel, and the two coincide.

Baselines. Primary: ShowUI-π (Hu et al., 2025) (SmolVLM-2 backbone with a flow-matching expert). Matched-pipeline discrete-tokenizer references: coord-bin (per-axis B=256) and raw-coord-BPE (Ours without the DCT stage). Three zero-shot single-call first-frame-only references: same-backbone SmolVLM-2-500M, GPT-4o, and Claude Sonnet 4.5 (Anthropic's drag tool). Prompting and API details in Appendix B.8.

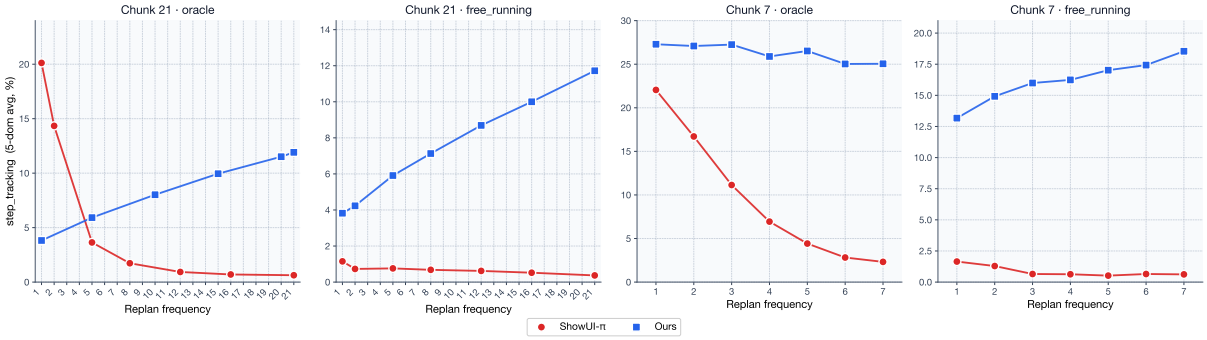

Figure 1: **Quality vs. replanning interval r on ScreenDrag.** Step-tracking accuracy at ϵ=20 px (5-domain average over n=505 episodes) for four panels: $H \in \{7, 21\} \times$ {oracle, free}. Ours peaks at r=H (full-chunk commit) under free-running; ShowUI-π peaks at r=1 under oracle (privileged per-step cursor injection) and collapses everywhere under free-running.

4.2 No-Replanning Results

Table 1 reports the no-replanning setting. Our policy at H=7×3 reaches 27.60% step-tracking under oracle prev. state and 19.76% free-running, against 0.63% for ShowUI-π at `replan_interval=21` and 0.03% for the same-backbone SmolVLM-2 language baseline. Any-endpoint accuracy on the same free-running row is 30.30% for Ours, against 10.87% for ShowUI-π at r=21 free and 12.47% at r=1 free. Under the deployable protocol, the chunked autoregressive head completes the drag at 2.4 to 2.8 times the rate of the flow-matching baseline.

Our 27.60% sits near Claude Sonnet 4.5's 26.52% single drag-tool call, a sanity check rather than a head-to-head tokenizer comparison.

A coord-bin baseline (per-axis B=256, H=7×3, matched pipeline) reaches 21.71% oracle but only 12.08% any-endpoint under free-running, against 30.30% for Ours at the same r=7. Encoding chunk-level shape through DCT correlation is more robust to compounding state error than independent per-axis quantization. The tokenizer at γ=80 reconstructs ScreenDrag near-losslessly (ATE=2.83 px, well below ϵ=20 px), so the residual gap to oracle comes from imperfect token prediction by the VLM, not from reconstruction. Tokenizer ablations and qualitative trajectories are in Appendices A and C.

4.3 Replanning Sweep

Figure 1 sweeps Ours and ShowUI-π across r for both $H \in \{7, 21\}$ and both rollout protocols. Under free-running, ShowUI-π stays below 1.5% everywhere; under oracle it peaks at 20.12% at r=1 and decays to 0.63% at r=21. Ours peaks at r=H under both protocols. Table 2 reports the H=21 trade-off as wall-clock ratios to our r=21 setting; absolute timings depend on engineering (KV-cache, batching, hardware). Appendix C shows representative trajectories at the two extremes (r=21 and r=1, free-running).

	Wall-	Step-track (oracle)		Step-track (free)	
r	clock	Ours	ShowUI-π	Ours	ShowUI-π
1	20.8×	3.82	20.12	3.82	1.15
2	10.8×	4.25	14.34	4.23	0.73
5	4.8×	5.92	3.64	5.91	0.76
8	3.0×	7.16	1.73	7.13	0.68
12	2.0×	8.84	0.93	8.69	0.62
16	2.0×	10.17	0.70	10.00	0.52
21	**1.0×**	**11.91**	0.63	**11.72**	0.37

Table 2: **Replan step vs. wall-clock vs. quality at H=21.** Step-tracking ↑ (ϵ=20 px); wall-clock ↓ as ratio to r=21 (ours). ♭Read off Fig. 1. ShowUI-π harness in App. B.

5 Conclusion

GUI drag is band-limited; a closed-form-decodable tokenizer adapted from FAST (Pertsch et al., 2025) fits the regime and feeds the VLM's autoregressive head under a single next-token cross-entropy loss. Under deployable free-running on ScreenDrag, our policy beats ShowUI-π at every replanning interval; ShowUI-π's reported strength rests on a privileged-state condition no deployed agent has. Scope is narrow (see Limitations); cross-platform benchmarks and adaptive chunk horizons are obvious next steps.

Limitations

Single-benchmark evaluation. We measure drag quality only on ScreenDrag (Hu et al., 2025), the benchmark released with our primary baseline. Transfer across operating systems, screen resolutions, and rendering toolkits is untested. A cross-platform drag benchmark would close this gap; we leave it to future work.

Released-checkpoint comparison to ShowUI-π. We evaluate ShowUI-π from its public checkpoint, not from a same-pipeline retrain on the ScreenDrag training split; the exact eval harness is documented in Appendix B. The headline gap therefore bundles tokenizer choice with possible differences in training-data mixture, grounding-data composition, optimizer schedule, and full-FT vs. adapter regime. Our coord-bin and raw-coord-BPE baselines (Table 1) are trained under matched conditions and isolate the action-representation effect inside our pipeline: they bound the policy-side contribution of our choice to the coord-bin-vs-Ours gap ($21.71 \rightarrow 27.60\%$ oracle, $12.08 \rightarrow 19.76\%$ free) and the raw-coord-BPE-vs-Ours gap ($20.10 \rightarrow 27.60\%$ oracle, $9.30 \rightarrow 19.76\%$ free), independent of the released-checkpoint confound. A same-pipeline retrain would resolve the remainder, but re-implementing the flow-matching expert exceeds our compute budget.

Dataset-fitted tokenizer. Our tokenizer's quantile-normalization percentiles and BPE merges are fit once on the training corpus. Trajectories far outside the training distribution (e.g., multi-monitor sweeps or sharp direction reversals) may quantize poorly. The rounding scale γ analytically bounds in-distribution reconstruction error, but says nothing about behavior under shifted action statistics. A calibration-set evaluation of quantile drift would expose this directly.

Acknowledgments

We thank the ScreenDrag and SmolVLM-2 maintainers for the publicly available data and backbone, and the FAST authors for the original tokenization recipe we adapt here for GUI drag. Anonymized acknowledgments to compute providers and reviewers will appear in the camera-ready version. We acknowledge the use of AI assistants for prose polishing; all research design, experimental work, analysis, and substantive writing decisions are the authors' own.

References

Nasir Ahmed, T. Natarajan, and K. R. Rao. 1974. Discrete cosine transform. *IEEE Transactions on Computers*, C-23(1):90–93.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, and 1 others. 2025. π_0: A vision-language-action flow model for general robot control. In *Robotics: Science and Systems (RSS)*.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, K. Choromanski, Tianli Ding, Danny Driess, Kumar Avinava Dubey, Chelsea Finn, Peter R. Florence, Chuyuan Fu, Montse Gonzalez Arenas, K. Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, A. Irpan, and 27 others. 2023. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *Conference on Robot Learning*.

Wentong Chen, Junbo Cui, Jinyi Hu, Yujia Qin, Junjie Fang, Yue Zhao, Chongyi Wang, Jun Liu, Guirong Chen, Yupeng Huo, Yuan Yao, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 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)*, pages 21936–21959, Vienna, Austria. Association for Computational Linguistics.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. 2024. Seeclick: Harnessing gui grounding for advanced visual gui agents. *Annual Meeting of the Association for Computational Linguistics*.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2024. Navigating the digital world as humans do: Universal visual grounding for gui agents. *International Conference on Learning Representations*.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. 2024. Cogagent: A visual language model for gui agents. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14281–14290. IEEE.

Siyuan Hu, Kevin Qinghong Lin, and Mike Zheng Shou. 2025. Showui-π: Flow-based generative models as GUI dexterous hands. *arXiv preprint arXiv:2512.24965*.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, A. Balakrishna, Suraj Nair, Rafael Rafailov, E. Foster, Grace Lam, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. 2024. Openvla: An open-source vision-language-action model. *Conference on Robot Learning*.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. 2025. Showui: One vision-language-action model for gui visual agent. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19498–19508. IEEE.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. 2023. Flow matching for generative modeling. In *ICLR*.

Andrés Marafioti, Orr Zohar, Miquel Farré, Merve Noyan, Elie Bakouch, Pedro Cuenca, Cyril Zakka, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, and 1 others. 2025. SmolVLM: Redefining small and efficient multimodal models. In *Conference on Language Modeling (COLM)*.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. 2025. FAST: Efficient action tokenization for vision-language-action models. In *Robotics: Science and Systems (RSS)*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1715–1725.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andrés Marafioti, and 1 others. 2025. SmolVLA: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*.

Jianwei Yang, Reuben Tan, Qianhui Wu, Ruijie Zheng, Baolin Peng, Yongyuan Liang, Yu Gu, Mu Cai, Seonghyeon Ye, Joel Jang, Yuquan Deng, and Jianfeng Gao. 2025. Magma: A foundation model for multimodal ai agents. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14203–14214. IEEE.

Keen You, Haotian Zhang, E. Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. 2024. Ferret-ui: Grounded mobile ui understanding with multimodal llms. *European Conference on Computer Vision*.

A Tokenizer-Level Ablation

The main text reports a policy-level numerical comparator for our tokenizer: the coord-bin policy in Table 1. That comparison varies several components at once (BPE vs. no BPE, per-axis vs. joint quantization, ACT/ep budget), so it does not isolate the DCT stage on its own. This appendix runs four *tokenizer-only* analyses on the trained Ours tokenizer (γ=80, vocab=2048, H=7) and on a control raw-coord-BPE tokenizer fit with identical hyperparameters but with the DCT stage removed. All four analyses use the ScreenDrag test split (n_{chunks}=1402 non-overlapping H=7 windows over 505 episodes, drawn at stride 7); the raw-coord-BPE control is fit on 35,866 training windows.

A.1 Tokenizer Hyperparameters and Fitting

All tokenizer parameters are fit *once* on the ScreenDrag training split; the 505-episode test set is held out from quantile estimation and BPE training. Quantile-normalization percentiles are q_{low}=0.01, q_{high}=0.99 over per-channel marginals; rounding scale γ=80, so the per-coefficient reconstruction bound $1/(2\gamma)$=1/160 in normalized coordinates lies well below the 20 px step-tracking tolerance at 1920×1080; BPE vocabulary $V \in \{2048, 4096, 8192\}$ (default V=2048); horizon $H \in \{7, 21\}$. The BPE merge table, tokenizer code, and training/evaluation harness will accompany the camera-ready revision and a public repository upon publication.

A.2 Spectral Concentration on ScreenDrag Trajectories

We compute the orthonormal Type-II DCT along the time axis of each H=7 chunk, square the coefficients, normalize per axis, and average over all test chunks. Table 3 reports the cumulative energy fraction $E_{\leq k} = (\sum_{i=0}^{k} |c_i|^2)/(\sum_{i=0}^{H-1} |c_i|^2)$ as a function of coefficient index.

k	btn	x	y
0 (DC)	0.932	0.883	0.870
≤ 1	0.954	0.986	0.985
≤ 2	0.972	0.988	0.992
≤ 3	0.986	0.991	0.994
≤ 4	0.995	0.997	0.998
≤ 5	0.999	0.998	0.999
≤ 6	1.000	1.000	1.000

Table 3: Cumulative DCT energy fraction by coefficient index for the three action dimensions on ScreenDrag test trajectories (n=1402 chunks, H=7). For the cursor coordinates, more than 98% of energy is in the first two coefficients; the high-frequency tail is small.

For the cursor coordinates, $\sim$ 88% of energy is in the DC component and a further $\sim$ 10% in the first AC coefficient. Higher-frequency coefficients each carry $\leq$ 1% on average. The button dimension is even more concentrated: 93% in DC, because most H=7 chunks sit entirely in a single

button state (fully pressed during a drag, or fully released between drags). This concentration is why the rounding stage can drop small high-frequency coefficients without measurable reconstruction error.

A.3 Per-Stage Sequence-Length Decomposition

We track the stream length through each stage of our pipeline (quantile-norm → DCT → scale-and-round → flatten → BPE) and compare with two ablation tokenizers: **coord-bin**, which discretizes each dimension to $B{=}256$ uniform bins and interleaves `[btn,`x`,`y`]` per timestep with no BPE; and **raw-coord-BPE**, identical to our pipeline but with the DCT stage removed. All three operate on the same $H{=}7$ chunks. Table 4 reports tokens per chunk and per $H{=}7{\times}3$ episode.

Variant	tok/chunk	tok/ep	ratio
raw floats	21	63	1.00×
coord-bin (no BPE)	21	63	1.00×
raw-coord-BPE	19.4	58.2	1.08×
Ours	**7.1**	**21.3**	**2.96×**

Table 4: Sequence-length decomposition across tokenizer variants on ScreenDrag test ($n{=}1402$ chunks, $H{=}7$, action dim 3). Mean tokens per chunk and per $H{=}7{\times}3$ episode. Ours per-chunk distribution: $p_{50}{=}7$, $p_{95}{=}14$, $\max{=}19$. The DCT stage reduces the stream by a factor of $2.74\times$ beyond what BPE alone achieves on raw quantized coordinates.

BPE applied directly to the rounded raw coordinates yields essentially no compression (19.4 vs. 21 tokens/chunk, $1.08\times$): the per-coordinate stream is high-entropy because each axis carries an independent two-byte symbol per timestep. After the DCT projects the same signal onto a basis where most energy is in two low-frequency coefficients, the rest of the stream becomes near-zero and BPE merges very long runs, taking the stream down to 7.09 tokens/chunk on average. The $2.74\times$ compression ratio between Ours and raw-coord-BPE comes specifically from the DCT stage, since γ, the BPE vocabulary, and the per-character alphabet are identical between the two variants.

A.4 Reconstruction Parity of Raw-Coord-BPE

To check that raw-coord-BPE is a fair non-DCT comparator (the shorter Ours stream should not buy compression by sacrificing reconstruction), we run raw-coord-BPE through the same closed-form autoencode loop used for Ours in §4.2. The tokenizer is fit on the same 35,866 training chunks with $\gamma{=}80$, vocab$=2048$. On the 1402 test chunks: ATE$=2.78$ px, Final@20$=1.00$, Any@20$=1.00$, against ATE$=2.83$ px for Ours at the same γ. Both tokenizers saturate the reconstruction floor on ScreenDrag, and the only remaining difference is token count.

A.5 BPE Merge Attribution

To confirm that the BPE stage actively merges the post-DCT stream rather than passing through base symbols, we tokenize each test chunk with the trained Ours tokenizer and count, per output token, the length of the underlying base-alphabet string it decompresses to. Across 9945 tokens generated from the 1402 test chunks: 14.8% are length-1 base-alphabet symbols, 85.2% are merges of length ≥ 2, 44.6% are merges of length ≥ 3, and 26.9% are merges of length ≥ 5. The pre-BPE stream is 21 base-alphabet characters per chunk; the post-BPE average is 7.09 tokens, which matches a $3\times$ effective merge length over the input.

A.6 Interpretation: DCT vs. BPE Contributions

Sections A.2–A.5 together show three things: the DCT stage is the dominant source of compression; the compression comes from the band-limited structure of ScreenDrag trajectories rather than from a BPE-training artifact; and it does not cost reconstruction quality. The downstream VLM accuracy at matched ACT/ep is in Table 1: at $H{=}7{\times}3$ raw-coord-BPE reaches $20.10/9.30\%$ (oracle/free), against $21.71/12.08\%$ for coord-bin and $27.60/19.76\%$ for our full pipeline. Two observations follow. First, BPE applied directly to scale-rounded raw coordinates does not improve over per-axis quantization, so BPE alone has little to merge on a high-entropy per-coordinate stream (matching the $1.08\times$ compression in Table 4). Second, the raw-coord-BPE-vs-Ours gap of $\approx$7.5 pp oracle and $\approx$10.5 pp free isolates the DCT stage's policy-level contribution: removing it inflates the per-episode stream by $2.74\times$ and the policy degrades by a comparable amount, in line with the known sensitivity of autoregressive policies to sequence length. The DCT stage is therefore necessary as well as sufficient, and BPE pays off only once the DCT stage has compressed the stream into a merge-amenable distribution.

B ShowUI-π Evaluation Harness

This appendix documents the exact harness used to produce every ShowUI-π number in Table 1, Table 2, and Figure 1. The $\sim$1.2% free-running and 0.63% matched-r=21 readings come from the same script that runs our policy; only the loaded checkpoint differs. The detail below is enough to reproduce all four published numbers (0.63% at r=H=21, 20.12% at r=1 oracle, $\sim$1.2% at r=1 free, and the intermediate-r readings in Figure 1) from the public checkpoint, without retraining.

B.1 Checkpoint and Policy Loading

We evaluate the public ShowUI-π checkpoint released with (Hu et al., 2025) as a `pretrained_model` directory in the LeRobot v3.0 format. The checkpoint ships a `config.json` (policy hyperparameters, including the SmolVLM-2-500M (Marafioti et al., 2025) backbone and the flow-matching action expert configuration) and a `train_config.json` (policy `type` field). Loading proceeds in three steps: (i) read the policy `type` from `config.json`, falling back to `train_config.json::policy.type`, and build a `policy_config` via LeRobot's `make_policy_config`; (ii) construct a `LeRobotDatasetMetadata("ScreenDrag", revision="v3.0")` pointing at the v3.0 parquet root to populate feature shapes (image $(3, 1080, 1920)$, state $(3,)$, action $(3,)$); and (iii) call `make_policy(cfg=policy_config, ds_meta=ds_meta)` followed by `make_pre_post_processors` with `pretrained_path` set to the checkpoint directory, so the same image normalization, state normalization, and action denormalization the checkpoint was trained with are restored verbatim. The policy is set to `eval()` mode; no weights are altered.

B.2 Benchmark Data and Image Preprocessing

For the oracle and matched-r free-running readings (the offline harness, Table 1) we use the HuggingFace dataset `h-siyuan/ScreenDrag` at LeRobot v3.0 (505 episodes $\times$ 21 steps; 5 domains $\times$ 101 episodes). The parquet rows carry one PNG screenshot per frame (encoded inside an `observation.images.screen` dict cell with a `bytes` key), the ground-truth previous state `observation.state` ($[btn, x, y]$ in screen pixels, sentinel $[0, -1, -1]$ at t=0 per the public release), the action `action` ($[btn, x, y]$ in pixels), and a `task_index` pointing into `meta/tasks.parquet` for the natural-language instruction.

Image preprocessing follows the released SmolVLA pipeline through the saved `preprocessor`: we decode the PNG with PIL, convert to RGB, cast to `uint8` numpy, permute to `(3, 1080, 1920)`, divide by 255 to land in $[0, 1]$, and prepend a batch dimension. The preprocessor then applies the checkpoint's stored mean/std and packs the input dict as the policy expects. We do not resize, crop, letterbox, or color-jitter: ScreenDrag is fixed at 1920$\times$1080 both at training and at evaluation, which is the resolution the ShowUI-π checkpoint was trained on.

B.3 State Conditioning: `oracle` vs. `free_running`

The two state-injection modes share the same rollout loop and differ only in how `observation.state` is constructed at each step. **Oracle prev. state.** At step t we read `observation.state` directly from the parquet row, which equals the ground-truth previous action at $t-1$, with sentinel $[0, -1, -1]$ at t=0. This is the protocol the ShowUI-π paper (Hu et al., 2025) reports its r=1 headline under and is what we label "oracle." At r=1 it is a privileged-state condition: per-step ground-truth cursor injection, no compounding error. **Free-running.** At step t we initialize `observation.state` to the sentinel $[0, -1, -1]$ at t=0 and overwrite it after each policy call with the policy's own predicted action $(\hat{btn}_{t-1}, \hat{x}_{t-1}, \hat{y}_{t-1})$, with $\hat{btn}$ thresholded at 0.5. Errors compound over the 21-step trajectory. This is the deployable protocol, what an executing agent would see at inference time, and what our headline comparisons against ShowUI-π run under.

Both modes call `policy.select_action(...)` once per step. Because the released ShowUI-π checkpoint is a chunked flow-matching expert that internally caches a future action queue, we explicitly clear `policy._queues["action"]` after every step so that the next call re-decodes a fresh chunk conditioned on the just-updated observation/state pair. Without this clear, the policy would silently execute the cached tail of the previous chunk and the harness would not implement step-by-step replanning.

B.4 Replanning Interval Implementation

The `replan_interval` r on $H{=}21$ controls how often the policy re-decodes. Concretely, the rollout loop maintains a counter `steps_since_decode`; at every step it queries `policy.select_action` (which re-decodes if and only if its internal action queue is empty), pops the first action from the returned chunk, and increments the counter. When the counter reaches r we clear `policy._queues["action"]` so the next step re-decodes a fresh H-step chunk under the current observation and state. This implements "decode a length-H chunk, execute its first r actions, re-decode" exactly. $r{=}1$ re-decodes every step (the published ShowUI-π headline protocol) and $r{=}H{=}21$ never re-decodes inside an episode (one forward pass per drag, the matched single-decode comparison with our policy).

For our policy we use the same replanning loop but call our autoregressive decode in place of `select_action`. The chunk-clear semantics are identical because our policy also emits a length-H chunk per decode.

B.5 Action Denormalization and Per-Step Scoring

The saved `postprocessor` maps the policy's normalized output back to screen pixels, producing a 3-vector $(\hat{btn}, \hat{x}, \hat{y})$. We clip $\hat{x}$ to $[0, 1919]$ and $\hat{y}$ to $[0, 1079]$ to enforce the screen bounds, and binarize $\hat{btn}$ at 0.5. The clipped pixel coordinates are scored against the parquet's ground-truth action $(x_t^{\text{gt}}, y_t^{\text{gt}})$ at step t: **step-tracking** at step t is the indicator $[\sqrt{(\hat{x}_t - x_t^{\text{gt}})^2 + (\hat{y}_t - y_t^{\text{gt}})^2} \le \epsilon]$ with $\epsilon{=}20$ px; **any-endpoint (Any@20)** is the indicator $[\min_t \sqrt{(\hat{x}_t - x_T^{\text{gt}})^2 + (\hat{y}_t - y_T^{\text{gt}})^2} \le \epsilon]$, where T is the episode endpoint; **ATE** is the mean over the episode of the per-step pixel distance $\sqrt{(\hat{x}_t - x_t^{\text{gt}})^2 + (\hat{y}_t - y_t^{\text{gt}})^2}$. Per-domain aggregates average over the 101 episodes in each of the five domains; the headline is the mean of the five per-domain rates (5-domain unweighted average) on the 505-episode test set.

B.6 From Rollouts to Published Numbers

All ShowUI-π readings in the paper come from a single deterministic aggregation over per-episode predictions. The harness writes `per_episode.json` (per-episode metrics) and `summary.json` (per-domain rates) per domain, and an `aggregate.json` with the 5-domain mean once all five complete. Concretely: the $\mathbf{0.37}\%$ at $r{=}H{=}21$ is the 5-domain mean of step-tracking on the released checkpoint, single chunk per drag, free-running rollout. It coincides with the oracle reading at the same r because at $r{=}H$ there is no internal boundary refresh and the two protocols differ only through evaluation-pipeline stochasticity (flow-matching is sampled, so per-run noise is ~ 0.05 pp; we hold the random seed fixed at 0 for reproducibility). The $\mathbf{20.12}\%$ at $r{=}1$ oracle is the same aggregation with per-step GT cursor injection (privileged state, see §B.3); this recovers the released-checkpoint headline to within sampling noise. The $\mathbf{1.15}\%$ at $r{=}1$ free-running is the same aggregation but with the policy's own predicted endpoint fed back as the next `observation.state`. Errors compound and the rate collapses by an order of magnitude relative to the oracle reading; the deterministic point estimate is 1.21%. Intermediate-r readings in Figure 1 use the same harness with `steps_since_decode`-based queue clears at the corresponding r.

No number in Table 1 or Table 2 requires retraining ShowUI-π; all are produced by loading the public checkpoint and varying only r, the state-injection mode, and (implicitly) which step's prediction is scored against which ground-truth coordinate. The released-checkpoint confound called out in the Limitations section therefore concerns training-time differences, not eval-time ones.

B.7 Reproducibility

The eval harness (offline parquet-driven and online data-driven replay) will be released with the camera-ready as a self-contained script that takes a pretrained checkpoint directory and a dataset root and reproduces the per-domain and 5-domain mean numbers under both state modes and any r. It depends only on `lerobot` v3.0+, `huggingface_hub`, `pandas`, `Pillow`, and `torch`; no proprietary or unreleased components are involved. Per-episode JSON outputs are also on disk for every cell in Tables 1 and 2 and will be uploaded alongside the harness.

B.8 Training, Inference, and Baseline Snapshots

Ours. SmolVLM-2-500M (`HuggingFaceTB/SmolVLM2-500M-Video-Instruct`), full fine-tune (no LoRA), bf16, `flash_attention_2`, AdamW 2×10^{-4}, batch

2/device, $8\times$H100, 10 epochs, $\sim$6 h/run. Inference on one H100, greedy decoding, batch 1.

Closed-source baseline snapshots. GPT-4o uses `gpt-4o-2024-08-06`. Claude Sonnet 4.5 uses `claude-sonnet-4-5-20250929` with the `computer_20250124` `left_click_drag` tool, screenshots pre-resized to 1280×720 with output coordinates scaled back.

C Qualitative Trajectory Comparison

Tables 1 and 2 quantify the ShowUI-π vs. Ours gap; this appendix shows what that gap looks like on individual ScreenDrag episodes. Figures 2 and 3 overlay each policy's predicted trajectory on the same three screenshots at the two extreme replanning intervals on $H{=}21$. Both figures use the deployable free-running protocol with no privileged state injection (these are the "free" columns of Table 2). The dashed green path is the ground-truth target trajectory; red traces are ShowUI-π; blue traces are Ours.

Together the two figures show why Ours peaks at $r{=}H$ rather than at $r{=}1$ (Figure 1): a single decode produces a chunk whose 21 predicted points are internally consistent with one realization of the low-frequency DCT coefficients, while per-step replanning forces each new chunk to start from a noisy own-prediction state and the cluster widens. ShowUI-π's free-running readings move in the opposite direction across r but stay an order of magnitude below Ours across the sweep.

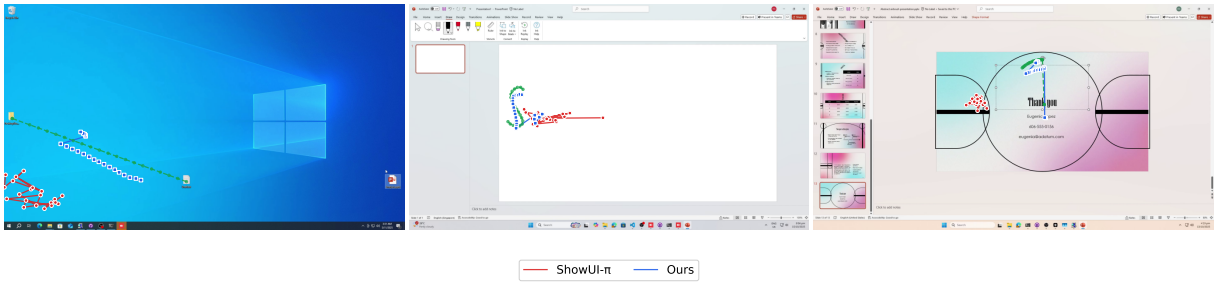

Figure 2: **Single-chunk rollouts** (r=21**, free-running,** H=21**).** Three representative ScreenDrag episodes. ShowUI-π commits the full 21-step chunk without any state refresh and its predicted trajectory escapes the target region (most visible in the middle panel as a long horizontal excursion away from the GT cluster). Ours stays inside the target cluster in every panel. Consistent with the r=21 free row of Table 2: Ours 11.72%, ShowUI-π 0.37%.

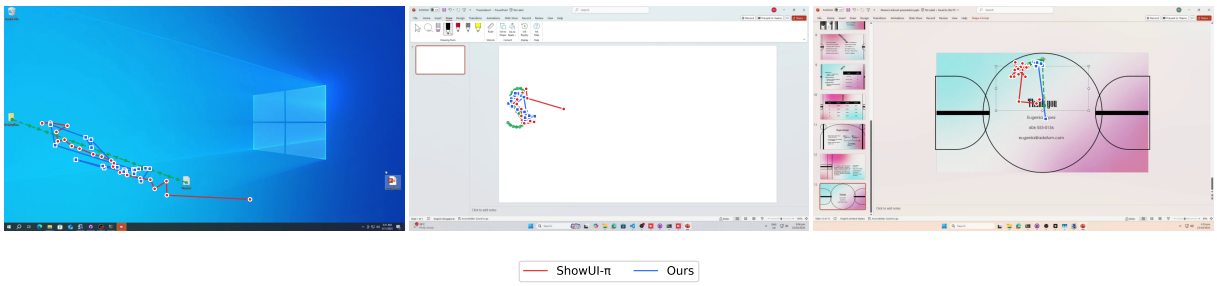

Figure 3: **Per-step replanning** (r=1**, free-running,** H=21**).** Same three episodes as Figure 2. Per-step re-decoding interrupts ShowUI-π's chunk-scale drift and its predictions cluster more tightly than at r=21, but they remain off the GT endpoint. Ours loses its r=21 tightness: every step now conditions on the policy's own predicted previous endpoint rather than committing a single internally consistent chunk, and the cluster widens. Consistent with Table 2: ShowUI-π rises 0.37% $\rightarrow$ 1.15% as r moves from 21 to 1, while Ours *drops* 11.72% $\rightarrow$ 3.82%.