ACTIONS ARE NOT TEXT: DIRECT EXECUTABLE-SET DECODING FOR GUI AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

GUI agents execute typed commands, yet many policies first serialize each action into text and then parse it back. We ask whether this round trip is necessary when the executable action set is finite and explicit. DirectAct retains the pretrained backbone and replaces action-string generation with a small head. This head selects a complete executable action without emitting action tokens. In an ecological comparison across Qwen, MAI, and Llama backbones, DirectAct achieves a 29.70–42.40× action-inference speedup over the evaluated JSON interface. We then compare against a compact LoRA-trained generator using greedy decoding over the same executable action sets and trained on the same data. We evaluate action accuracy on 1,218 prospectively collected states from 88 successful AndroidWorld reference workflows. DirectAct passes the predeclared non-inferiority test on Qwen and MAI with a two-percentage-point margin. Across all three backbones, DirectAct remains 4.51–5.13× faster even though the generator emits only about six to seven tokens. These results support a division of work: a pretrained backbone contextualizes the state, while a small head directly selects the executable action.

1 INTRODUCTION

GUI agents interact with an environment through structured commands, such as clicking a control or entering a value. Many policies generate each action as text or JSON, which the executor parses and binds to the current UI. When the available controls and values already define a finite executable action set, the policy could instead select a member of that set directly. Generating its textual representation adds sequential decoding steps before execution.

This setting arises naturally in structured GUI systems. Accessibility trees and DOM-based interfaces expose addressable controls (Rawles et al., 2025; Zheng et al., 2024), while task instructions or maintained state supply candidate values. Together with the executor's supported operations, these fields can define complete executable actions. The policy must still determine which executable action advances the task. Our study begins once the candidates are available and examines how the policy selects among them. We therefore ask whether direct selection can reduce action inference latency while preserving the accuracy of an autoregressive generator trained to choose from the same executable action set.

Our method, DirectAct, uses a frozen backbone to contextualize the structured state and its candidates in one forward pass. A small factorized head scores complete executable actions and selects the highest-scoring action, emitting no action tokens. DirectAct compares complete executable candidates rather than committing to successive tokens in an action string. Figure 1 contrasts this interface with autoregressive action generation.

We evaluate JSON-based action generation, used by AndroidWorld's M3A agent (Rawles et al., 2025), as an ecological reference point. Across Qwen, MAI, and Llama, DirectAct achieves a 29.70–42.40× action-inference speedup over the evaluated JSON interface. This comparison measures the practical difference between the two configurations; the controlled comparison asks whether the advantage survives specialized training and output compression.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

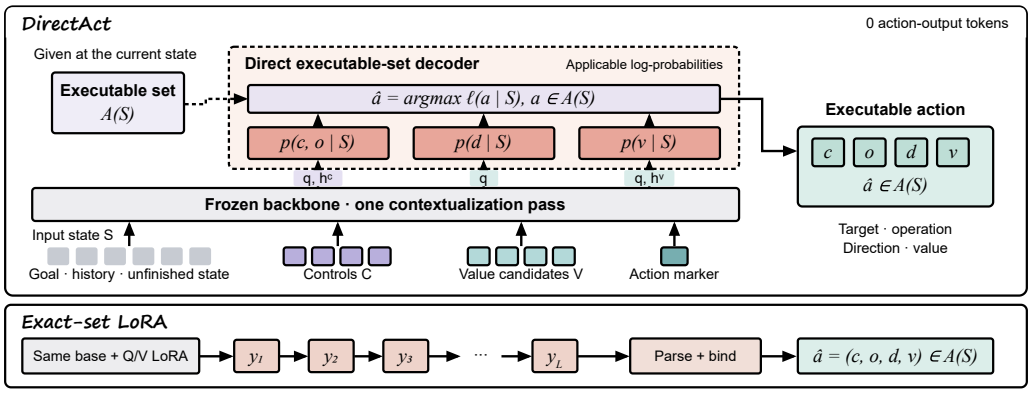


Figure 1: DirectAct overview. A frozen backbone contextualizes the structured state. A factorized head scores complete executable actions, and a global argmax selects one from $\mathcal{A}(S)$, without generating action tokens. Greedy exact-set LoRA instead serializes an action through trie-constrained decoding over the same executable set.

Compact formats shorten the generated sequence, and constrained decoding keeps outputs legal, but both retain token-by-token generation. Our trained baseline, greedy exact-set LoRA, generates compact action strings constrained to the same executable set as DirectAct. The two methods use the same training data and input, with similar trainable parameter counts.

We evaluate 1,218 states from 88 newly collected, successful AndroidWorld reference workflows. On these states, DirectAct passes the predeclared action-level non-inferiority test on Qwen and MAI. An additional Llama evaluation extends the comparison to a third backbone. Across all three, DirectAct remains 4.51–5.13× faster despite the generator emitting only about six to seven tokens.

Together, these results identify the action output interface as a practical optimization target, even after task-specific training and output compression.

This paper makes three contributions:

1. We introduce DirectAct, which pairs a frozen backbone with a small factorized head to select complete actions from an observed executable set without generating action tokens.
2. We demonstrate a 29.70–42.40× action-inference speedup over the evaluated JSON interface in an ecological comparison across three backbones.
3. We show that DirectAct remains 4.51–5.13× faster than trained, compact greedy exact-set LoRA under the matched protocol, even though the generator emits only about six to seven tokens. DirectAct also meets the predeclared action-accuracy non-inferiority criterion on Qwen and MAI.

2 Related Work

GUI action modeling. GUI-Actor uses an attention-based action head for coordinate-free GUI grounding (Wu et al., 2026). CoCo-Agent decomposes smartphone commands into an action type and a conditional target, while retaining an autoregressive output interface (Ma et al., 2024). GUI-Libra reweights action and grounding tokens during GUI-specific post-training (Yang et al., 2026). MAGA similarly reallocates distillation signal according to structured action correctness (Yan et al., 2026). These generation-based methods motivate evaluating a task-trained generator alongside the JSON interface.

Compact and latent action representations. AgentCPM-GUI introduces a compact generated action space explicitly to reduce output length and mobile execution latency (Zhang et al., 2025). Latent Action Reparameterization learns latent units corresponding to multi-step semantic behaviors, shortening the effective decision horizon (Zeng et al., 2026). Our study asks whether direct selection can further reduce latency when a generator already uses a compact representation of the same complete current action.

Constrained structured generation. Grammar-constrained decoding can guarantee syntactic membership in a structured output language and can make the grammar input-dependent (Geng et al., 2023). Automata-based decoding further addresses the mismatch between formal languages and subword tokenization (Koo et al., 2024). Such methods remove format failures, but an autoregressive model still traverses the output token sequence. Our trained baseline uses a prefix trie over tokenized actions from the current executable action set. This allows us to compare inference cost when both methods produce executable actions, while the generator retains sequential decoding.

Removing generation from the GUI critical path. AAPT precomputes conditional policy trees during idle periods and routes transient events to prepared branches, explicitly removing autoregressive decoding from the decision-time critical path (Dong et al., 2026). ActionEngine crawls applications into a state-machine memory and synthesizes executable programs, with visual regrounding on failure (Zhong et al., 2026). The recently released Jev (Almeida, 2026) similarly produces typed decisions without autoregressive text generation.

We investigate whether selecting complete executable GUI actions directly can reduce inference latency while preserving action accuracy. Our evaluation first establishes the gains over an ordinary JSON interface, then tests whether they persist against a task-trained compact generator matched on training data, inputs, and executable action sets.

3 Problem Formulation

3.1 Structured state and executable actions

At step t, let S_t denote the available state: the user goal, compact history, unfinished task state, and current structured GUI observation. Let $C_t = \{c_i\}_{i=1}^{n_t}$ be current controls augmented with a null-control symbol. Thus n_t includes the null-control candidate. Let O and D be finite operator and direction vocabularies, respectively. The current value candidates are $V_t = \{v_j\}_{j=1}^{m_t}$. An action is the tuple $a_t = (c_t, o_t, d_t, v_t)$, where direction and value may be null when inapplicable. For each operator o, $C_t(o)$, $D(o)$, and $V_t(o)$ denote the control, direction, and value domains allowed by its signature, using a null symbol for an unused field. These domains define the input-dependent set of well-typed actions

$$\mathcal{T}(S_t) = \{(c, o, d, v) : o \in O,\ c \in C_t(o),\ d \in D(o),\ v \in V_t(o)\}.$$

Here `tap`, `type`, and `set_range` require a visible control. The terminal operator requires the null control, and `scroll` requires a non-null direction. Direction or value is canonicalized to null when the operator does not consume it.

An action that satisfies the type constraints may still be unavailable in the current state. Let $M_t(c, o, d, v) = 1$ when the state builder declares that complete tuple executable in the current state. This predicate specifies which actions the executor accepts, not whether they advance the task. The explicit executable set is

$$\mathcal{A}(S_t) = \{a \in \mathcal{T}(S_t) : M_t(a) = 1\}. \tag{1}$$

The state builder supplies $\mathcal{A}(S_t)$ as part of the interface state. The matched comparison evaluates action selection over this same set.

3.2 Autoregressive action generation

An autoregressive policy generates an action as a sequence of L tokens, each conditioned on the state and preceding tokens. The sequence is then parsed and its fields bound to controls and values in the current state.

This path has the following approximate computational cost, including parsing and binding:

$$T_{\mathrm{AR}} \approx T_{\mathrm{prefill}} + \sum_{j=2}^{L} T_{\mathrm{decode},j} + T_{\mathrm{parse}} + T_{\mathrm{bind}}.$$

Here T_{prefill} denotes input contextualization, and $T_{\mathrm{decode},j}$ is the cost of the subsequent decoding step for token j. Prefill produces the first-token logits, so generating L tokens requires $L - 1$ subsequent decoding steps. T_{parse} and T_{bind} account for parsing the generated text and resolving its fields.

Compact formats reduce L, but generating the remaining tokens still requires sequential decoding. This motivates examining how the action output interface affects inference latency when the executable candidates are already explicit.

4 METHOD

DirectAct replaces action-string generation with direct selection from $\mathcal{A}(S_t)$. A frozen backbone contextualizes the state and candidates, and a small factorized head scores complete executable actions. The highest-scoring action is returned for execution.

4.1 SHARED CONTEXTUALIZATION

The backbone stays frozen so that task-specific learning is confined to the action head. The input lists the goal, unfinished state, recent executed actions, tagged current controls, tagged value candidates, and a shared action marker. The backbone processes this sequence in one forward pass. We take the final-layer hidden state at the last token of each control or value span as h_i^c or h_j^v, respectively. A dedicated null-control span is encoded in the same way. The action query q is the final-layer hidden state at the last token of the shared model input. This token follows all candidates and can attend to the full input sequence. These query and candidate features are reused across all tuple scores, so selecting among complete actions requires no additional backbone pass.

4.2 SCORING AND SELECTING COMPLETE ACTIONS

We factorize scoring to reuse control/operator, direction, and value scores across actions, but select a complete tuple to respect the compatibility constraints in $\mathcal{A}(S_t)$. Thus the component heads share computation without making independent field decisions. The head computes control/operator logits $z_{i,o}$ from q and each h_i^c. It also computes direction logits from q and value logits from q and each h_j^v. A joint softmax over control/operator logits and separate softmaxes over directions and values yield $p_\theta(c, o \mid S_t)$, $p_\theta(d \mid S_t)$, and $p_\theta(v \mid S_t)$. Appendix C.1 gives the network equations. Each complete action receives the sum of its applicable component log probabilities:

$$\begin{aligned}\ell_\theta(a \mid S_t) = \log p_\theta(c, o \mid S_t) + \mathbf{1}[o = \texttt{scroll}] \log p_\theta(d \mid S_t) \\ + \mathbf{1}[o \in O_v] \log p_\theta(v \mid S_t),\end{aligned} \tag{2}$$

where $O_v = \{\texttt{type}, \texttt{key}, \texttt{set_range}\}$. The score ℓ_θ ranks complete actions; it is not a log probability normalized over $\mathcal{A}(S_t)$. DirectAct selects the best complete tuple from the executable action set in Equation (1):

$$\hat{a}_t = \arg \max_{a \in \mathcal{A}(S_t)} \ell_\theta(a \mid S_t). \tag{3}$$

Inference enumerates this set, computes the tuple scores, and returns the maximizing action. Because different action types use different subsets of the component scores, their normalization terms need not cancel when comparing tuples. We therefore use normalized log probabilities rather than summing raw logits. No action tokens are generated.

The resulting inference path has approximate cost

$$T_{\text{direct}} \approx T_{\text{prefill}} + T_{\text{head}}.$$

Here T_{head} includes component scoring, executable-action enumeration, and the final argmax. These operations reuse the contextualized features without subsequent token-decoding steps.

4.3 TRAINING OBJECTIVE

GUI trees may expose multiple controls that execute the same reference action. Training should reward their combined probability rather than require an arbitrary single control. If $\mathcal{Y}_t$ is the set of equivalent positive control/operator pairs, the joint loss is multi-positive:

$$\mathcal{L}_{c,o} = -\log \frac{\sum_{(i,o) \in \mathcal{Y}_t} \exp z_{i,o}}{\sum_{k=1}^{n_t} \sum_{o' \in O} \exp z_{k,o'}}.$$

Direction uses cross-entropy on every training row, including the `none` label when no direction is needed. Value cross-entropy is included when a reference value is represented in V_t, restricting value supervision to choices the head can actually select. Each row's loss is the mean of its available supervised terms. An unavailable value target leaves control/operator and direction supervision intact. At inference, the direction score contributes only to scroll actions, as specified in Equation (2).

5 Experimental Design

The controlled experiment uses the same training examples, input, candidate order, pretrained backbone weights, and executable action sets. DirectAct performs a complete-tuple argmax; LoRA learns a short tuple serialization and selects from the same executable action set autoregressively.

5.1 Equal data and input

Both interfaces use the same 675 rows, including state, history, controls, values, candidate order, and action marker. DirectAct trains the action head; the generator applies LoRA to the query and value projections in the frozen language layers. The generator outputs four space-separated fields: control index, operator, direction, and value index, using none for inapplicable fields. It receives no extra visual or state input. The four-field action format also retains a null value symbol for training rows whose reference value is absent from V_t; those value targets are masked during training.

The prospectively collected dataset records the complete executable candidates as interface state rather than inferring the set from reference labels. At evaluation time, DirectAct and exact-set LoRA receive this same recorded executable set $\mathcal{A}(S_t)$. DirectAct applies the complete-tuple argmax above. LoRA serializes exactly those tuples with the four-field action format, tokenizes them as in supervised fine-tuning, and constructs a trie. Greedy generation masks tokens outside the trie, guaranteeing membership in $\mathcal{A}(S_t)$ while retaining one autoregressive step per output token.

The two methods have similar trainable parameter counts (Table 3). The same 28 examples with unavailable value targets mask value supervision for both interfaces.

Optimization follows each parameterization: DirectAct uses 200 cached-feature updates and LoRA uses nine passes (765 updates) through the same rows. The nine-epoch LoRA schedule was chosen using historical development evaluations. Appendix C reports the complete training configuration.

5.2 Evaluation data and accuracy assessment

We fix both heads, both nine-epoch LoRAs, the four-field action format, a two-percentage-point margin, and the analysis before prospective collection. The collector runs official AndroidWorld initialization and terminal scoring over new dynamic instances. Failed or infrastructure-invalid workflows remain recorded, while only officially successful reference trajectories provide action labels. The final evaluation set contains 1,218 supported actions from 88 successful workflows spanning seven task families. The 21 additional attempted workflows are retained outside the labeled evaluation set. No reference action falls outside its recorded executable set. This predeclared protocol applies to Qwen and MAI; Llama was evaluated after inspection of the evaluation results as an exploratory extension.

Let $R_{wi} \subseteq \mathcal{A}(S_{wi})$ contain the recorded reference tuple and any tuple declared execution-equivalent by the state builder. Equivalence covers duplicate GUI controls or value candidates that dispatch the same environment action. Null fields are canonicalized according to the operator signature. Here w indexes workflows and i indexes the evaluated actions within each workflow; $N = 1{,}218$ is the total number of evaluated actions. For policy P, action accuracy is

$$\mathrm{Acc}(P) = \frac{1}{N} \sum_{w,i} \mathbf{1}[\hat{a}^P_{wi} \in R_{wi}].$$

Thus correctness is exact at the complete executable-action level while avoiding an arbitrary penalty for duplicate interface representations.

The primary quantity of interest is the action-weighted accuracy difference, estimated by $\hat{\Delta} = \mathrm{Acc}(\mathrm{Direct}) - \mathrm{Acc}(\mathrm{LoRA})$. Each of 20,000 bootstrap replicates samples whole workflows with

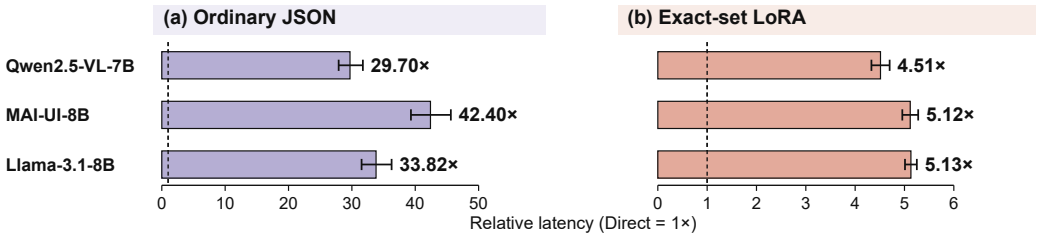


Figure 2: Action inference latency relative to paired DirectAct timings on 200 states. Whiskers show paired-bootstrap 95% intervals. (a) Ecological JSON comparison with different prompts and training status. (b) Comparison with greedy exact-set LoRA using matched training data, inputs, and executable action sets. Note the different axis scales; the Llama evaluation is exploratory.

replacement and keeps all actions within every sampled workflow. DirectAct is non-inferior when the one-sided 95% lower bound satisfies

$$\mathrm{LCB}_{.95}(\hat{\Delta}) > -0.02. \tag{4}$$

5.3 Matched inference timing

Timing uses 200 evenly spaced states from the 1,218-state evaluation set. Each path receives two warm-up calls. Qwen and MAI then rotate each state through DirectAct, greedy exact-set LoRA, and unconstrained LoRA generation within one loaded model. Llama alternates DirectAct and greedy exact-set LoRA on the same selected states. Inputs are identical across compared paths within each backbone. We measure action inference latency from tokenized input to action output, with the model already loaded. DirectAct timing includes the full backbone forward pass, executable-action scoring, and argmax; LoRA timing includes prefill and greedy generation. Greedy exact-set LoRA constrains every token to a serialization of a tuple in the executable action set. The unconstrained baseline measures the contribution of the Python trie callback to latency.

Each backbone runs separately on one NVIDIA GH200 at batch size one. Autoregressive policies use KV caching, and every call is bracketed by CUDA synchronization. Detailed timing boundaries and runtime settings appear in Appendix D.2.

5.4 Ecological JSON comparison

We separately evaluate the JSON interface. It prompts the pretrained backbone without task-specific fine-tuning, using a fixed prompt and unconstrained decoding. Accuracy uses the same 1,218 states; latency alternates it with DirectAct on the 200-row timing subset. The interfaces retain their respective prompts and training configurations, defining the ecological comparison. Appendix G gives the full protocol and results.

6 Results

We first report the practical cost of JSON-based action generation, then use the matched comparison to test whether the gains persist after training and output compression.

6.1 The cost of generating actions as JSON

Across Qwen, MAI, and Llama, DirectAct achieves a 29.70–42.40× action-inference speedup over the evaluated JSON interface. This interface emits 70.53–94.78 tokens (Figure 2a). The JSON interface reaches 33.33%, 36.70%, and 23.07% action accuracy, respectively. Accuracy conditional on successful parsing remains 24.25–42.33%, so malformed JSON accounts for only part of its error. This practical cost motivates the matched comparison below, which tests what remains after adaptation and output compression. Full results and paired timing intervals appear in Appendix G.

Table 1: Action accuracy under the matched comparison with greedy exact-set LoRA. Direct denotes DirectAct in tables and figures. LCB is the one-sided 95% lower confidence bound from whole-workflow bootstrap. The predeclared non-inferiority margin is -2 pp for Qwen and MAI.

Backbone	Interface	Correct	Accuracy	Δ (pp)	LCB (pp)
Qwen2.5-VL-7B	Direct	928/1218	76.19%	+11.33	+8.53
	Greedy exact-set LoRA	790/1218	64.86%	—	—
MAI-UI-8B	Direct	918/1218	75.37%	+1.56	+0.08
	Greedy exact-set LoRA	899/1218	73.81%	—	—
Llama-3.1-8B	Direct	903/1218	74.14%	+23.48	+21.36
	Greedy exact-set LoRA	617/1218	50.66%	—	—

6.2 Action quality is retained without action-token generation

DirectAct retains action-selection quality while removing the sequential output path. Table 1 reports accuracy and non-inferiority statistics for the comparison over the same executable action set; Figure 2b shows the corresponding controlled latency comparison.

DirectAct passes the predeclared action-weighted non-inferiority test (Equation (4)) on Qwen and MAI, with higher point accuracy on both. The one-sided 95% lower bounds clear the -2-point margin (Table 1). Every constrained LoRA output is executable, so this comparison measures selection accuracy within the shared executable set.

Llama-3.1-8B-Instruct extends the same comparison to an independent model family. It reproduces the interface result under the same training rows, evaluation states, input, and executable action sets. Exhaustive sequence scoring in Appendix B recovers much of the Qwen and Llama accuracy gap, showing that greedy token decisions contribute to the observed errors. DirectAct compares complete executable actions directly and achieves similar point accuracy at substantially lower latency than the evaluated exhaustive scorer.

6.3 Six action tokens still incur about fivefold latency

With full input contextualization included for both interfaces, DirectAct remains 4.51–5.13$\times$ faster than greedy exact-set LoRA across the three backbones. The generator emits only 6.20–6.68 action tokens, whereas DirectAct emits none. Figure 2b shows the paired-bootstrap 95% intervals; Appendix D reports the per-backbone timings and intervals.

Unconstrained LoRA, without the Python trie callback, still takes 4.02$\times$ DirectAct's latency on Qwen and 4.73$\times$ on MAI. The measured latency advantage therefore persists even when generation incurs no trie-constraint overhead.

7 Discussion

7.1 Action interfaces as a computational design choice

For finite, explicitly enumerated actions, the output interface is a computational design choice, not just a formatting convention. The JSON comparison shows the practical latency benefit of DirectAct over the evaluated JSON interface. DirectAct also meets the predeclared action-accuracy criterion against trained exact-set LoRA on Qwen and MAI.

Both DirectAct and exact-set LoRA return complete executable actions from the same candidate set. They differ in how the model makes that choice: the generator commits to successive tokens, whereas DirectAct scores complete actions after a shared backbone pass. Compact encoding shortens the token sequence, and constrained decoding ensures executable outputs, but neither removes sequential decoding. The matched results show that these improvements leave a substantial latency gap. Removing the trie callback preserves that gap, indicating that it is not specific to the constraint implementation. For explicit action sets, improving the text interface and replacing it are therefore materially different optimization strategies.

The recent release of Jev (Almeida, 2026) reflects interest in replacing text generation with typed decisions. Its public introduction outlines a new architecture, parallel sampler, and training approach, but does not explain in detail how the model is constructed and trained. Our study examines what can be achieved by changing the action interface around an existing pretrained backbone. DirectAct retains the backbone for state contextualization and trains a lightweight head for complete-action selection. The resulting speedups, together with the matched accuracy results, support a practical alternative to reducing backbone size: retain the model's contextual computation and remove the sequential generation used to express its action.

7.2 Separating state understanding from action selection

The resulting design criterion is whether the policy must select an available action or construct a new one. When complete executable actions are already enumerated, direct selection is a natural alternative to text generation. Function and tool interfaces could admit the same design when all typed arguments are enumerated, not merely the function names. Free-form messages, code, and values outside the candidate set instead require generation or an additional mechanism for constructing candidates. This suggests a broader architectural principle: use generation where new content is needed, and consider direct selection where the remaining decision is among explicit executable alternatives.

8 Limitations

The predeclared non-inferiority analysis covers Qwen and MAI; the Llama evaluation provides exploratory evidence across model families. The matched comparison aligns training data, inputs, executable action sets, and approximate trainable parameter counts. Training objectives differ by parameterization, and training compute was not matched. The evaluation covers seven AndroidWorld task families. Transfer to web and desktop interfaces remains to be evaluated.

9 Conclusion

When a GUI action is already a finite executable choice, serializing it is optional. In the JSON comparison, JSON-based action generation takes 29.70–42.40× DirectAct's action inference latency. Under matched data, input, parameter scale, and executable action sets, DirectAct emits zero action tokens. It remains 4.51–5.13× faster across three backbones under this matched protocol. It also passes the predeclared action-level non-inferiority test against greedy exact-set LoRA on Qwen and MAI. The backbone contextualizes the state, and the head selects the action directly. This division supports direct selection among finite executable actions, reserving generation for open-ended outputs.

Reproducibility statement

The accompanying anonymous repository contains method implementations, training and evaluation code, ordered input records, per-state predictions, and paired timing records. An offline verification script recomputes the reported accuracy, latency ratios, and bootstrap intervals from these records. Appendix A describes the data records; Appendix C gives the training configuration, Appendix D the timing settings, and Appendix E the input construction. Code and data are available at https://anonymous.4open.science/r/actions-are-not-text-551D/README.md.

AI use statement

Generative AI tools were used for giving feedback on experimental design, assisting with implementation, and polishing the manuscript.

References

Diogo Almeida. Introducing System One Models & Jev. TypeSafe AI Blog, September 2026. URL https://typesafe.ai/blog/introducing-system-one-models-and-jev.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. pp. 9313–9332, 2024.

Zihan Dong, Rui Qian, Qishi Zhan, Dongshen Peng, Kaixin Li, and Yu Li. Why are gui agents correct but late? decode on the decision-time critical path, tested with pre-compiled policy trees. *arXiv preprint arXiv:2607.28399*, 2026.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured nlp tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 10932–10952, 2023.

Terry Koo, Frederick Liu, and Luheng He. Automata-based constraints for language model decoding. *arXiv preprint arXiv:2407.08103*, 2024.

Xinbei Ma, Zhuosheng Zhang, and Hai Zhao. Coco-agent: A comprehensive cognitive mllm agent for smartphone gui automation. pp. 9097–9110, 2024.

Chris Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. 2025:406–441, 2025.

Qianhui Wu, Kanzhi Cheng, Rui Yang, Chaoyun Zhang, Jianwei Yang, Huiqiang Jiang, Jian Mu, Baolin Peng, Bo Qiao, Reuben Tan, et al. Gui-actor: Coordinate-free visual grounding for gui agents. *Advances in Neural Information Processing Systems*, 38:15101–15128, 2026.

Hang Yan, Zhangxuan Gu, Beitong Zhou, Jiaxuan Chen, Runze Li, Yusong Hu, Shuheng Shen, and Changhua Meng. Maga: Multi-platform self-fusion of gui agents via structured action distillation. *arXiv preprint arXiv:2607.29320*, 2026.

Rui Yang, Qianhui Wu, Zhaoyang Wang, Hanyang Chen, Ke Yang, Hao Cheng, Huaxiu Yao, Baolin Peng, Huan Zhang, Jianfeng Gao, et al. Gui-libra: Training native gui agents to reason and act with action-aware supervision and partially verifiable rl. *arXiv preprint arXiv:2602.22190*, 2026.

Qingwen Zeng, Wenhao Huang, Zerui Xu, Zijie Guo, Yu Sun, Cheng Yang, Siru Ouyang, Jiri Gesi, Fang Wu, Jiayi Zhang, et al. Latent action reparameterization for efficient agent inference. 2026.

Zhong Zhang, Yaxi Lu, Yikun Fu, Yupeng Huo, Shenzhi Yang, Yesai Wu, Han Si, Xin Cong, Haotian Chen, Yankai Lin, et al. Agentcpm-gui: Building mobile-use agents with reinforcement fine-tuning. pp. 155–180, 2025.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*, 2024.

Hongbin Zhong, Fazle Faisal, Luis França, Tanakorn Leesatapornwongsa, Adriana Szekeres, Kexin Rong, and Suman Nath. Actionengine: From reactive to programmatic gui agents via state machine memory. *arXiv preprint arXiv:2602.20502*, 2026.

Hanzhang Zhou, Xu Zhang, Panrong Tong, Jianan Zhang, Liangyu Chen, Quyu Kong, Chenglin Cai, Chen Liu, Yue Wang, Jingren Zhou, et al. Mai-ui technical report: Real-world centric foundation gui agents. *arXiv preprint arXiv:2512.22047*, 2025.

A Reproducibility and Data Records

This appendix describes the training, development, and evaluation records in the anonymous repository. Training and bootstrap random seeds are retained in the code.

Training and development data. Both trained interfaces use the same 675 training rows, including augmented inputs with rotated control order and recomputed features. The repository also retains 35 historical diagnostic rows that are included in the final training set, and a separate 68-state early evaluation set. The latter later formed part of a 143-state historical development evaluation used to choose the nine-epoch LoRA schedule. The reported heads use the final update of a fixed 200-update run, and the adapters use the final update of the nine-epoch run.

Prospective evaluation. The main evaluation contains 1,218 states from 88 successful reference workflows; the records also retain the status of all 109 attempted workflows. Its state and workflow identifiers are disjoint from the training and earlier development records. All seven evaluated task families occur in the training data, so this comparison evaluates new instances within those families. Table 6 gives the family counts. The Qwen and MAI checkpoints and analysis were fixed before collection. Accuracy intervals resample whole workflows, whereas timing intervals resample paired states, each with 20,000 replicates. The matched and JSON timing records retain their respective DirectAct measurements.

B Exhaustive Sequence Scoring

The primary exact-set LoRA policy follows a greedy path through the trie of executable actions. To assess the effect of greedy search, we keep each trained checkpoint and executable action set fixed and score every complete action serialization. This selects the maximum a posteriori (MAP) action within the executable set under the sequence model. Let $y_j(a)$ be its jth token and $L(a)$ its length, including the final EOS token; $y_{<j}(a)$ denotes the preceding tokens. The sequence score and selected action are

$$s_\phi(a \mid S) = \sum_{j=1}^{L(a)} \log p_\phi(y_j(a) \mid S, y_{<j}(a)), \qquad \hat{a}_{\mathrm{MAP}} = \arg\max_{a \in \mathcal{A}(S)} s_\phi(a \mid S).$$

We apply no length normalization. A state contains 16.83 executable actions on average and at most 31. Table 2 reports this exploratory accuracy comparison on all 1,218 states and implementation latency on the same 200 states used by the primary timing study.

Table 2: Exploratory comparison using exhaustive sequence scoring. Accuracy uses all 1,218 states. Latency is a descriptive cross-run measurement on the aligned 200-state subset for the evaluated exhaustive implementation. Ratios compare it with DirectAct and greedy exact-set LoRA.

	Action accuracy			Exhaustive latency		
Backbone	Greedy	Exhaustive	Direct	Mean (ms)	/ Direct	/ Greedy
Qwen2.5-VL-7B	64.86%	76.68%	76.19%	476.44	7.90×	1.75×
MAI-UI-8B	73.81%	74.06%	75.37%	593.10	7.91×	1.55×
Llama-3.1-8B	50.66%	70.85%	74.14%	514.23	8.12×	1.58×

Exhaustive scoring closes the Qwen accuracy gap and recovers most of the Llama gap; MAI changes little. DirectAct is 0.49 points below exhaustive scoring on Qwen and 1.31 and 3.28 points above it on MAI and Llama. The improvement under exhaustive scoring identifies a contribution from greedy search. DirectAct reaches similar point accuracy by scoring complete actions directly.

The exhaustive implementation is slower than both alternatives, taking 7.90–8.12× DirectAct latency and 1.55–1.75× greedy latency. The exhaustive scorer records per-state latency in separate runs over the full evaluation set. We extract the same 200 states used by the primary timing study to compute the reported means and latency ratios. The implementation repeats the prompt across candidate batches. These measurements characterize the evaluated exhaustive scorer rather than a lower bound on optimized sequence scoring.

C Training and Inference Configuration

The following details specify the DirectAct head and the training and decoding settings used in the matched comparison with exact-set LoRA.

C.1 HEAD PARAMETERIZATION

The head jointly scores each control/operator pair. With width $r = 128$,

$$u_i = \mathrm{GELU}(W_q\ \mathrm{LN}(q) + W_c\ \mathrm{LN}(h_i^c)),$$
$$z_{i,o} = w_o^\top u_i.$$

Direction is scored from q. Each value is scored by a learned function of the action query and the candidate value representation:

$$z^d = \mathrm{MLP}_d(\mathrm{LN}(q)),$$
$$z_j^v = w_v^\top\ \mathrm{GELU}(U_q\ \mathrm{LN}(q) + U_v\ \mathrm{LN}(h_j^v)).$$

Here LN denotes layer normalization without learned affine parameters. All linear layers include biases, omitted from the notation above. The direction MLP has two linear layers with hidden width r and a GELU activation between them. A joint softmax over control/operator logits and separate softmaxes over directions and values yield the probabilities in Equation (2).

C.2 OPTIMIZATION AND DECODING SETTINGS

Table 3 details the comparison with similar trainable parameter counts across all three reported backbones. Relative to DirectAct's parameter count, the differences are 3.84% on Qwen and 8.67% on MAI (Zhou et al., 2025). Both methods use the same 675 rows and candidate order. DirectAct extracts the final hidden states once with the frozen backbone and stores them in FP16. For head training, it casts these features to FP32 and accumulates losses over all 675 rows before each update. LoRA trains through the BF16 backbone with batch size one and eight-step gradient accumulation. Neither method uses a learning-rate schedule. Table 3 reports the optimization settings for each interface.

Table 3: Training configuration for all three reported backbones. "Full" means one optimizer step after accumulating all 675 DirectAct examples.

	Qwen		MAI		Llama	
Setting	Direct	LoRA	Direct	LoRA	Direct	LoRA
Training actions	675	675	675	675	675	675
Backbone weights	frozen	frozen	frozen	frozen	frozen	frozen
Head width / LoRA rank	128	7	128	5	128	6
LoRA α / dropout	—	14 / 0.05	—	10 / 0.05	—	12 / 0.05
LoRA target modules	—	`q_proj`, `v_proj`	—	`q_proj`, `v_proj`	—	`q_proj`, `v_proj`
Trainable parameters	2,295,948	2,207,744	2,623,628	2,396,160	2,623,628	2,555,904
Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
Learning rate	10^{-3}	2×10^{-4}	10^{-3}	2×10^{-4}	10^{-3}	2×10^{-4}
Weight decay	10^{-4}	10^{-2}	10^{-4}	10^{-2}	10^{-4}	10^{-2}
Epochs	200	9	200	9	200	9
Accumulation	full	8	full	8	full	8
Optimizer updates	200	765	200	765	200	765
Maximum prompt tokens	1,536	1,536	1,536	1,536	1,536	1,536

Each training row uses the mean of its available control–operator, direction, and value losses, with a log-sum-exp multi-positive term for execution-equivalent controls. Direction supervision includes the `none` label on non-directional actions. Twenty-eight rows lack a represented value target; value loss is masked for those rows in both methods. LoRA uses teacher-forced token cross-entropy on the minimal four-field target. It also uses gradient checkpointing, SDPA attention, gradient norm clipping at 1.0, and a maximum of eight target tokens. DirectAct clips its gradient norm at 5.0. The primary LoRA inference rule is token-wise greedy traversal of a trie containing exactly the tokenizations of the supplied executable actions. Exact-set LoRA uses a 32-token cap and appends EOS to each action serialization in its trie. DirectAct applies one global argmax over its tuple scores.

C.3 ACTION SELECTION AND COMPUTATIONAL COST

Equation (3) compares complete tuples in $\mathcal{A}(S)$, so its output belongs to that set whenever it is nonempty. Although the score in Equation (2) adds component log probabilities, the fields are not

selected independently. When $\mathcal{A}(S)$ is not a Cartesian product, its constraints couple the choices. Applying a softmax over these tuple scores would leave the maximizing action unchanged and is unnecessary for selection.

For an action of length $L(a)$, prefill produces the first-token logits and autoregressive generation requires $L(a) - 1$ subsequent decoder invocations. DirectAct instead uses one head evaluation followed by an argmax over the executable set. The measured 4.51–5.13× speedup reflects this difference together with hardware parallelism and batching in the evaluated implementations.

D Inference Timing Results and Configuration

D.1 Matched timing results

Table 4 gives the absolute timings, output-token counts, and paired bootstrap intervals underlying Figure 2b. The intervals resample paired states. The unconstrained LoRA control is evaluated on Qwen and MAI to assess the contribution of the Python trie callback.

Table 4: Matched action inference timing on 200 evenly spaced evaluation states. Latency ratios are relative to DirectAct; 95% intervals use paired bootstrap resampling of states.

Backbone	Interface	Mean ms	Latency/Direct	95% interval	Out. tok.
Qwen2.5-VL-7B	Direct	60.35	1.00×	—	0
	Greedy exact-set LoRA	272.38	4.51×	[4.33, 4.70]	6.56
	Unconstrained LoRA	242.33	4.02×	[3.83, 4.21]	6.61
MAI-UI-8B	Direct	75.00	1.00×	—	0
	Greedy exact-set LoRA	383.72	5.12×	[4.96, 5.28]	6.68
	Unconstrained LoRA	355.04	4.73×	[4.57, 4.91]	6.77
Llama-3.1-8B	Direct	63.35	1.00×	—	0
	Greedy exact-set LoRA	325.01	5.13×	[5.01, 5.25]	6.20

D.2 Measurement configuration

Timing excludes tokenization, environment dispatch, observation stabilization, upstream state acquisition, and model loading.

The matched and JSON timing studies use the same runtime configuration but different interleaved comparisons. Each study runs each backbone in a separate process on one GH200 GPU. The matched study rotates three methods on Qwen and MAI and alternates two on Llama. The JSON study alternates DirectAct and the JSON interface. Across the 200 matched timing states, each path processes 139,730 input tokens on Qwen and MAI and 138,224 on Llama. Inputs are identical across compared paths within each backbone.

Table 5: Settings for the inference timing measurements. The JSON interface is timed in a separate two-way interleaved run with a newly measured DirectAct path.

Setting	Direct	Greedy exact-set LoRA	JSON interface
GPU allocation	1× GH200 per backbone task	1× GH200 per backbone task	1× GH200 per backbone task
Backbone precision	BF16	BF16	BF16
Direct head / tuple scoring	FP32	—	—
Attention implementation	PyTorch SDPA	PyTorch SDPA	PyTorch SDPA
KV cache	Off (`use_cache=False`)	On (`use_cache=True`)	On (`use_cache=True`)
Inference batch size	1	1	1
Warm-up	2 calls per path	2 calls per path	2 calls per path
Timing barrier	CUDA sync before and after	CUDA sync before and after	CUDA sync before and after
Qwen/MAI matched order	Three-way rotation	Three-way rotation	—
Llama matched order	Two-way alternation	Two-way alternation	—
JSON comparison order	Two-way alternation	—	Two-way alternation
Runtime	Containerized PyTorch (aarch64)	Same	Same

The runtime row describes the container environment. Individual Python, PyTorch, CUDA, Transformers, and PEFT version strings were not recorded in the timing logs.

E Input Construction and an Executable-Action Example

We describe how the recorded state and executor candidates form the complete action set shared by both interfaces, then illustrate how an indexed tuple maps to an executable command.

E.1 Candidate construction

The state builder preserves the order of visible controls and the executor's candidate actions. It extracts field–value requirements from the goal and updates them using executed history. Explicit executor values absent from the remaining requirements are appended to the value candidates. Candidate values retain their associated record and field role. Entries with the same record identifier, field role, and literal value are deduplicated while preserving their order. Each supported executor candidate is then mapped to a control, operator, direction, and value index. An action with a value argument is included when that argument matches a represented value; duplicate value positions yield the corresponding alternative tuples. Inapplicable fields are canonicalized, and duplicate tuples are removed. This construction uses the state and executor candidates; reference actions are used afterward to assign evaluation labels. Execution-equivalent controls or value positions are accepted as alternative reference tuples.

E.2 Shared input prefix

DirectAct and exact-set LoRA receive the goal, remaining field–value requirements, the last three executed actions, and visible controls. Tagged control and value candidates follow in their recorded order, ending with an action marker. Qwen and MAI tokenize this prefix directly; Llama wraps it in its native chat template. The query q is taken at the final action-marker token for Qwen and MAI and at the final assistant-prompt token for Llama. The 1,536-token budget includes any chat-template tokens. All tagged candidates and the action marker are retained; if necessary, tokens are removed from the beginning of the preceding context.

E.3 An action with a value argument

One recorded single-expense entry state requests an expense named `Workshops`, with amount `121.2`, category `Education`, and note `A need`. These are value candidates 0–3, respectively. Visible controls include `Name` at index 2, `Amount` at index 3, and `Note` at index 10. The executor offers typing the corresponding literal values into these fields. They become the executable tuples $(2, \texttt{type}, \texttt{none}, 0)$, $(3, \texttt{type}, \texttt{none}, 1)$, and $(10, \texttt{type}, \texttt{none}, 3)$ within the full 18-action set. The reference selects the first tuple, entering `Workshops` into `Name`. DirectAct scores all complete candidates in the set, whereas exact-set LoRA selects through a trie of their serializations, including `2 type none 0`. Binding the selected value index retrieves the literal string supplied to the executor. Thus both interfaces select the control and its argument, not just an action type.

F Detailed Accuracy Results

Breaking down the aggregate comparison shows how the accuracy difference between DirectAct and exact-set LoRA varies across task families and action characteristics.

F.1 Accuracy by task family

Table 6 reports descriptive, post-hoc action accuracies for all seven task families across the three backbones.

The gains vary across task families. Qwen's greedy gap is concentrated in Clock and Recipe actions, whereas greedy exact-set LoRA is better on SMS reply and single-Expense actions. MAI is closer overall, with a large DirectAct gain on Clock, a smaller gain on multi-Recipe, and LoRA advantages

Table 6: Descriptive per-family action accuracy (%) on the same evaluation set. N is shared across backbones; Δ is DirectAct minus greedy exact-set LoRA in percentage points.

		Qwen			MAI			Llama		
Task family	N	Direct	LoRA	Δ	Direct	LoRA	Δ	Direct	LoRA	Δ
Clock timer	118	83.90	55.08	+28.81	65.25	44.92	+20.34	55.08	37.29	+17.80
Expense, multiple	250	86.00	85.20	+0.80	87.20	87.60	−0.40	87.60	74.40	+13.20
Expense, single	97	88.66	94.85	−6.19	90.72	93.81	−3.09	89.69	76.29	+13.40
Recipe, multiple	468	69.23	50.21	+19.02	68.59	64.74	+3.85	67.95	37.82	+30.13
Recipe, single	157	69.43	47.77	+21.66	66.88	68.79	−1.91	69.43	39.49	+29.94
SMS reply	52	46.15	75.00	−28.85	76.92	100.00	−23.08	73.08	51.92	+21.15
SMS send	76	93.42	93.42	0.00	90.79	96.05	−5.26	88.16	61.84	+26.32
All	1,218	76.19	64.86	+11.33	75.37	73.81	+1.56	74.14	50.66	+23.48

on several other families. With Llama, DirectAct exceeds greedy exact-set LoRA in all seven families, with its largest gaps on Recipe actions.

F.2 Accuracy by action characteristics

We examine how the accuracy difference varies with value requirements, executable action set size, and action type in Table 7.

Table 7: Descriptive accuracy differences by action characteristics across three backbones. Entries are $\mathrm{Acc}(\mathrm{Direct}) - \mathrm{Acc}(\mathrm{LoRA})$ in percentage points; N is shared across backbones. The blocks for value requirements and executable action set size each partition all 1,218 states. The operator block covers the 1,130 states with nonterminal reference actions.

Action group	N	Qwen	MAI	Llama
Value required	759	+18.84	+6.32	+28.85
No value required	459	−1.09	−6.32	+14.60
1–8 executable actions	132	+0.76	−0.76	+3.03
9–16 executable actions	513	−2.53	−3.70	+14.04
$\geq$ 17 executable actions	573	+26.18	+6.81	+36.65
`type`	460	+25.43	+6.09	+43.04
`tap`	337	−4.75	−7.12	+16.32
`key`	299	+8.70	+6.69	+7.02
`scroll`	34	+11.76	−14.71	+73.53

All three backbones show larger DirectAct gains on value-requiring actions and states with at least 17 executable actions. The remaining pattern varies by backbone: Llama also gains on taps and actions without values. In contrast, Qwen and MAI favor greedy exact-set LoRA on those groups. Together with the exhaustive-scoring analysis, these results show where the two decision rules differ across backbones.

G Ecological JSON Comparison

G.1 Results and comparison settings

The JSON interface receives the task intent, remaining field–value requirements, recent actions, and indexed controls, then generates a JSON action. It uses the pretrained backbone without task-specific fine-tuning, a fixed prompt, and unconstrained decoding. Accuracy is evaluated on all 1,218 states. Latency uses the same 200 evenly spaced states as the controlled timing study and alternates DirectAct and the JSON interface within one loaded model. The JSON interface receives fewer input tokens than DirectAct in this study; both paths retain their established input representation.

Table 8: JSON comparison on the same evaluation set. Latency is interleaved on 200 evenly spaced states. DirectAct is remeasured alongside the JSON interface. Its absolute mean may therefore differ slightly from Table 4. Each ratio uses the DirectAct measurement from the same timing run.

Backbone	Interface	Accuracy	Parse	Mean ms	Lat./Direct	Out. tok.
Qwen2.5-VL-7B	Direct	76.19%	—	63.59	1.00×	0
	JSON interface	33.33%	99.92%	1888.76	29.70×	70.53
MAI-UI-8B	Direct	75.37%	—	72.99	1.00×	0
	JSON interface	36.70%	86.70%	3094.75	42.40×	94.78
Llama-3.1-8B	Direct	74.14%	—	62.98	1.00×	0
	JSON interface	23.07%	95.16%	2129.56	33.82×	89.08

DirectAct lowers mean action inference latency by 96.63–97.64%. Paired-bootstrap 95% intervals for the latency ratio are [27.91, 31.73] on Qwen, [39.32, 45.62] on MAI, and [31.53, 36.27] on Llama. The JSON interface's accuracies conditional on a parsed output are 33.36%, 42.33%, and 24.25%, respectively. This comparison quantifies the cost of the ordinary JSON interface; different prefixes and training status prevent a causal attribution of its full accuracy gap to serialization.

G.2 Decoding settings

The backbone's chat template wraps the JSON-interface input as one user message and adds its generation prompt. Inputs are left-truncated to 1,536 tokens. Decoding is greedy with KV caching, a 192-token output cap, and the tokenizer's EOS stopping token. The output uses literal argument values, whereas exact-set LoRA emits indices into the shared candidate list.

H Closed-Loop Feasibility Study

We evaluate live executability in AndroidWorld (Rawles et al., 2025) using a MAI head that conditions value selection on the chosen control. A controller tracks the required field–value updates. This development variant uses a different checkpoint from the main offline study and selects values conditionally rather than scoring complete actions directly. The controller marks a required update as complete when the next observation confirms the corresponding field–value change. Equivalent accessibility-tree elements map to the same executable action.

On five generated single-expense entry development instances, DirectAct with MAI completes 5/5 tasks and all 20 required field–value updates. The unadapted compact MAI interface completes 0/5 tasks and 4/20 updates. Mean episode time is 85.36 versus 160.11 seconds, and total measured model inference is 2.20 versus 185.56 seconds across these five instances. These development instances demonstrate live executability of this variant on single-expense entry tasks.

The reported aggregates use the ten completed episodes. One compact-interface attempt was interrupted when its model-service allocation ended; its completed retry supplies that instance's result. The interrupted record is retained separately and excluded from the reported means.