Visual Token Routing: Learning to keep tokens that matter

Pranjal Aggarwal **Sean Welleck**
Carnegie Mellon University

Abstract

Computer-use agents (CUAs) have recently achieved superhuman performance on several long-horizon benchmarks. However, their computational cost limits their use in long-horizon tasks and real-time domains such as video games. A key bottleneck is representing each screenshot as thousands of visual tokens, which accumulate into long contexts as agents reason over extended interaction histories. We observe that most of these visual tokens are either semantically irrelevant to the agent's next action or temporally redundant across screenshots. Building on this observation, we introduce Visual Token Routing (VTR), a joint training framework for learning sparse visual representations that preserve the information needed for action prediction. In VTR, we train the vision-language model together with a lightweight spatio-temporal router that examines each image region in the current and previous screenshots and decides whether its visual token enters the language model. We optimize the router using an adaptive loss to keep the number of visual tokens as small as possible while minimizing performance degradation. With just 2,000 training steps, VTR matches the performance of the full-resolution model on OSWorld, CUA-World, and video game action prediction, while using 2.5-5.8$\times$ fewer visual tokens. These savings reduce inference cost by up to 3.3$\times$ and speed up prefill by up to 2.9$\times$. VTR also outperforms lowering the image resolution and other baselines for dropping tokens, by up to 12.8 points on OSWorld. Overall, VTR enables computer-use agents to process only the visual tokens they need, allowing them to be used in long-horizon tasks and real-time domains at a fraction of the computational cost.[1]

1 Introduction

Computer-use agents (CUAs), which operate software by observing screenshots and issuing mouse and keyboard actions, have recently achieved superhuman performance on several long-horizon benchmarks [Xie et al., 2024, Aggarwal et al., 2026b,a]. However, their computational cost limits their use in long-horizon tasks and real-time domains such as video games [Aggarwal et al., 2026a, Abhyankar et al., 2026]. A key bottleneck is the representation of each screenshot as thousands of visual tokens, which accumulate into long contexts as agents reason over extended histories of interaction. For instance, Qwen3.5 [Qwen Team, 2026] encodes a single 1080p screenshot into about 2,000 visual tokens, so an agent that keeps its full history on a 100-step task has over 200,000 visual tokens in its context by the final step. In video games, agents observe the screen many times per second, and even at a reduced resolution of 640×360, one minute of play at 30 frames per second amounts to about 400,000 visual tokens. Furthermore, in real-time environments, speed is the primary bottleneck for frontier agents [Aggarwal and Welleck, 2026].

Existing methods reduce the number of visual tokens by keeping the tokens that receive the most attention within each image [Chen et al., 2024, Yang et al., 2025], or by dropping tokens that remain

[1] Code and models are available at `https://pranjal2041.github.io/vtr`.

Preprint.

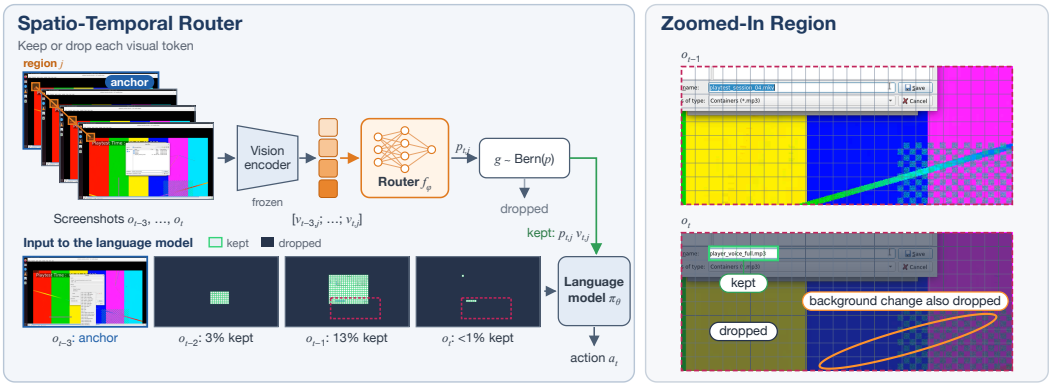


Figure 1: **Overview of Visual Token Routing (VTR). Spatio-temporal router:** A lightweight router reads each image region across the current and previous screenshots and decides whether its visual token enters the language model. **Zoomed-in region:** While the agent types a file name into a save dialog in front of a playing video, the router keeps only the tokens of the typed file name. It drops the regions that did not change, and also drops the circled part of the video, which changed but is not relevant to the task.

unchanged across video frames [Shen et al., 2025, Bagrov et al., 2025]. However, these hand-designed rules keep a fixed fraction of tokens or apply a fixed threshold to every input, regardless of what the agent needs for its next action. Other work learns which visual tokens to keep jointly with the model for single images in general visual question answering [Huang et al., 2025, Ye et al., 2025]. However, computer-use agents face fundamentally new challenges, such as long histories of nearly identical screenshots, and actions that depend on small regions of the screen, like a cursor or a single spreadsheet cell.

We observe that most visual tokens in an agent's history are either semantically irrelevant to its next action or temporally redundant across screenshots. For instance, the taskbar and the empty parts of a document rarely affect the agent's next click, and consecutive game frames share most of their background. Building on this observation, we introduce Visual Token Routing (VTR), a joint training framework for learning sparse visual representations that preserve the information needed for action prediction (§2; Figure 1). In VTR, a lightweight spatio-temporal router processes each image region across the current and previous screenshots and outputs a decision to keep the visual token or not. Dropped tokens are removed from the input sequence, so the language model processes only the kept tokens, which retain their original positional encodings.

We train the router jointly with the language model to satisfy two objectives: (1) keeping as few visual tokens as possible, and (2) preserving the model's accuracy in predicting the agent's next action. However, training the router poses two challenges. First, dropping a token is a discrete decision that provides no gradient to the router, which we address by scaling each kept token by its keep probability, as in mixture-of-experts models [Shazeer et al., 2017, Fedus et al., 2022]. Second, unlike mixture-of-experts models, where each token is processed by a fixed number of experts, the router must also decide how many tokens to keep. We therefore measure how much the screenshots help the model predict each action, using reference models trained with and without them, and adaptively weight the two objectives so that the model keeps this benefit while dropping tokens. In doing so, the router learns how many visual tokens each input needs while preserving the model's performance, rather than relying on a fixed token budget.

We use VTR to train vision-language models on computer-use trajectories and video game play, and evaluate them on OSWorld [Xie et al., 2024], CUA-World [Aggarwal et al., 2026b], and video game action prediction. With just 2,000 training steps, VTR matches the performance of the full-resolution model in all three domains, while using 2.5-5.8$\times$ fewer visual tokens (Figure 4). These savings reduce inference cost by up to 3.3$\times$ and speed up prefill by up to 2.9$\times$ (Figure 5). In addition, VTR outperforms lowering the image resolution and hand-designed rules for dropping tokens, by up to 12.8 points on OSWorld. Furthermore, we find that learning visual token selection jointly with the model is crucial, improving performance by 13.2 points on OSWorld over the same router trained separately from the model.

In summary, our contributions are three-fold. First, we introduce Visual Token Routing (VTR), a joint training framework for learning sparse visual representations, in which a lightweight spatio-temporal router decides which visual tokens enter the language model. Second, we propose a dual objective that trains the router to keep as few visual tokens as possible while preserving action prediction, so that the router learns how many visual tokens each input needs without a fixed token budget. Third, through extensive evaluations, we show that VTR matches full-resolution performance on OSWorld, CUA-World, and video game action prediction with 2.5-5.8$\times$ fewer visual tokens and up to 3.3$\times$ lower inference cost, that it outperforms other token reduction approaches, and that learning visual token selection jointly with the model is crucial.

2 Visual Token Routing

2.1 Problem Setup

Our goal is to reduce the number of visual tokens that a computer-use agent processes at each step, without degrading the actions that it takes. Next, we describe how such an agent observes and acts, and then formalize our method, Visual Token Routing (VTR).

Computer-use agents. A computer-use agent performs a task specified by a natural-language instruction p over multiple steps. At each step t, it receives an observation o_t, such as a screenshot of a desktop or a frame of a video game, and takes an action a_t, such as a mouse click or a key press. The agent chooses a_t based on its interaction history $h_t = (p, o_1, a_1, \ldots, o_{t-1}, a_{t-1}, o_t)$, and we model it as a vision-language model π_θ that predicts $a_t \sim \pi_\theta(\cdot \mid h_t)$.

Visual tokens. The vision encoder of π_θ maps each observation o_τ to N visual tokens $v_{\tau,1}, \ldots, v_{\tau,N} \in \mathbb{R}^D$, one per image region, where D is the hidden size of the language model. For instance, $N = 2{,}040$ for a 1080p screenshot in Qwen3.5-2B [Qwen Team, 2026]. As each observation in h_t contributes N tokens, the language model processes tN visual tokens at step t.

Visual token routing. A router assigns each visual token a gate $g_{\tau,j} \in \{0, 1\}$, and the language model receives the routed history $\tilde{h}_t$, which contains $v_{\tau,j}$ only if $g_{\tau,j} = 1$, along with all text and action tokens. Dropped tokens are removed from the input sequence, so the language model does not process them. Our goal is to minimize the fraction of kept visual tokens,

$$R_t = \frac{1}{tN} \sum_{\tau=1}^{t} \sum_{j=1}^{N} g_{\tau,j}, \tag{1}$$

without degrading the prediction of a_t compared with the full history h_t. We do not fix R_t in advance, so the number of kept tokens can vary across observations and steps. However, learning the gates raises two challenges. First, the gates are discrete, so the training loss provides no gradient for choosing them. Second, the right trade-off between keeping fewer tokens and predicting accurately differs across inputs. Next, we describe VTR, which addresses both.

2.2 Spatio-Temporal Router

Whether a visual token is redundant depends on the earlier observations in the history. We therefore base each routing decision on the same image region across the current and the K previous observations. Formally, given visual token $v_{\tau,j}$, the router concatenates the visual tokens at the same position j,

$$x_{\tau,j} = [v_{\tau-K,j}; \ldots; v_{\tau-1,j}; v_{\tau,j}], \tag{2}$$

and predicts a keep probability $p_{\tau,j} = \sigma(f_\phi(x_{\tau,j}))$, where f_ϕ is a small MLP. Beyond recognizing regions that have not changed, the router learns which regions matter for the agent's actions through the objective in §2.3.

Gating. The router samples a gate $g_{\tau,j} \sim \mathrm{Bernoulli}(p_{\tau,j})$, and a kept token enters the language model scaled by its keep probability,

$$\tilde{v}_{\tau,j} = p_{\tau,j}\, v_{\tau,j} \quad \text{if } g_{\tau,j} = 1. \tag{3}$$

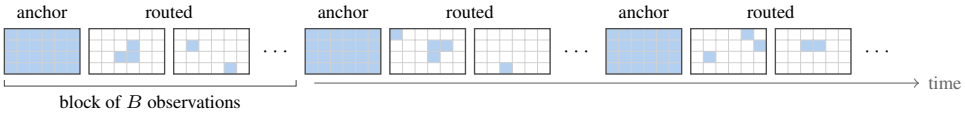


Figure 2: **Where the router acts.** The first observation of each block is an anchor and is kept in full, while the router keeps only some regions, shown shaded, of the other observations in the block.

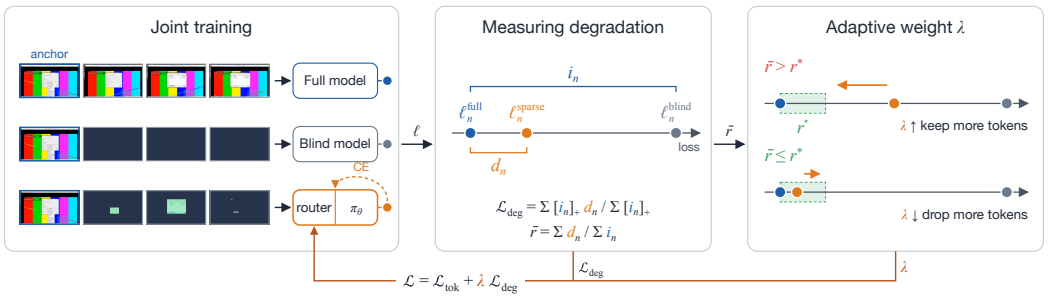


Figure 3: **Training in VTR. Joint training:** We train π_θ and the router together with the cross-entropy (CE) loss on the kept tokens, and also train the router to keep fewer tokens with $\mathcal{L} = \mathcal{L}_{\text{tok}} + \lambda\, \mathcal{L}_{\text{deg}}$. The full and blind models see all tokens and only the anchors, and are used only during training. **Measuring degradation:** We compare the losses of the three models to measure how much the full history helps (i_n) and how much dropping tokens hurts (d_n). **Adaptive weight λ:** We increase λ when the share of this help that π_θ loses, $\bar{r}$, exceeds the small tolerance $r^\star$, so that the router keeps more tokens, and decrease it otherwise.

The scaling makes the language model's loss depend on $p_{\tau,j}$, which gives the router a gradient despite the discrete gate, as in mixture-of-experts models [Shazeer et al., 2017, Fedus et al., 2022]. Kept tokens retain their original positions, so the language model still knows where each kept region lies on the screen. At inference, the router runs inside vLLM [Kwon et al., 2023] and samples the gates as in training.

Anchors. We group observations into blocks of B consecutive observations and call the first observation of each block an anchor (Figure 2). Anchors are kept in full, and the router acts only on the other observations in a block, with its history starting at the anchor. The anchor gives the model a full view of the screen to compare later observations against, and each new block refreshes this view, so regions that the router keeps dropping are never out of date for long.

Sliding router. We also consider a sliding variant, in which only the first observation is kept in full, and the router's history is a rolling window of the K previous observations without block boundaries. This variant avoids the cost of periodic anchors and uses the same trained router.

2.3 Training

Training the router requires balancing two goals: keeping as few visual tokens as possible and preserving the accuracy of action prediction (Figure 3). A natural objective is a weighted sum of the two,

$$\mathcal{L} = \mathcal{L}_{\text{tok}} + \lambda\, \mathcal{L}_{\text{deg}}, \tag{4}$$

where $\mathcal{L}_{\text{tok}}$ is the expected fraction of kept tokens, $\mathcal{L}_{\text{deg}}$ measures how much dropping tokens hurts action prediction, and λ is a weight. This raises two questions: how to measure $\mathcal{L}_{\text{deg}}$, and how to set λ. For the first question, the loss of π_θ alone is not enough, since some actions are hard to predict even from the full history, and others can be predicted from the anchors alone. We therefore compare π_θ against two reference models: one that sees the full history and one that sees only the anchors. For the second question, the right λ is unknown in advance, since a large λ keeps too many tokens and a small λ loses too much accuracy. We therefore set a target for how much accuracy the router may give up, measured as a fraction of how much the full history improves over the anchors alone, and adapt λ during training to meet it.

Measuring degradation. We call the two reference models the full model and the blind model. For each action token n, let ℓ_n^{full}, ℓ_n^{blind} and ℓ_n^{sparse} be the cross-entropy losses of the full model, the blind model and π_θ given only the kept tokens. We define

$$i_n = \ell_n^{\text{blind}} - \ell_n^{\text{full}}, \qquad d_n = \ell_n^{\text{sparse}} - \ell_n^{\text{full}}, \tag{5}$$

so i_n is how much the full history helps predict token n, and d_n is how much dropping tokens hurts it. Subtracting ℓ_n^{full} removes how hard token n is to predict in the first place. We then average d_n over action tokens, weighted by i_n,

$$\mathcal{L}_{\text{deg}} = \frac{\sum_n [i_n]_+ \, d_n}{\sum_n [i_n]_+}, \tag{6}$$

where $[x]_+ = \max(x, 0)$. Tokens that the anchors alone predict as well as the full history get zero weight, since dropping tokens should not matter for them, while tokens that depend most on the full history get the most weight. Finally, we compute $\mathcal{L}_{\text{tok}}$ as the mean keep probability over the visual tokens outside the anchors, which, unlike the gates, is differentiable.

Adaptive weight. Rather than fixing λ, we formulate training as a constrained optimization problem: minimize the expected fraction of kept tokens, subject to the model losing at most a fraction $r^\star$ of the improvement that the full history provides over the anchors,

$$\min \ \mathcal{L}_{\text{tok}} \quad \text{subject to} \quad \bar{r} \le r^\star, \qquad \text{where} \quad \bar{r} = \frac{\sum_n d_n}{\sum_n i_n}. \tag{7}$$

Eq. (4) corresponds to the Lagrangian of this problem, with λ as the Lagrange multiplier. We therefore update λ by dual ascent [Boyd and Vandenberghe, 2004], alternating between a gradient step on the router and an update of λ,

$$\lambda \leftarrow \max\big(0, \ \lambda + \alpha \, (\bar{r} - r^\star)\big), \tag{8}$$

where $\bar{r}$ is computed over the training batch and α is the step size. In practice, λ increases when $\bar{r}$ exceeds $r^\star$, placing more weight on $\mathcal{L}_{\text{deg}}$ so that the router keeps more tokens, and decreases when $\bar{r}$ is below $r^\star$, so that the router drops more tokens. At a fixed point of this procedure, either $\bar{r} = r^\star$ or $\lambda = 0$, as required for a solution of Eq. (7). Notably, $r^\star$ constrains the degradation rather than the number of kept tokens, so the router learns how many tokens each input needs.

Joint training. Since π_θ was pre-trained on full images, we train it jointly with the router so that it learns to predict actions from the kept tokens alone. At each training step, we sample gates for each example and update π_θ and the router with the cross-entropy loss of π_θ given only the kept tokens. We then sample a second set of gates and update only the router with Eq. (4), keeping π_θ fixed. The full and blind models are initialized from the same model as π_θ and trained with the cross-entropy loss on the same examples, so that i_n and d_n reflect differences in the input rather than in training. At inference, we use only π_θ and the router, so the reference models add no cost.

Initialization. In joint training, the router learns which tokens matter only through the loss of π_θ, which gives a weak signal when the router starts from random weights. We therefore first train the router alone to predict simple labels based on §1: a token is labeled as kept only if its region changed from the previous observation, so that it is not temporally redundant, and belongs to an object segmented by SAM [Kirillov et al., 2023], so that it is likely semantically relevant. Joint training then starts from this router and refines its decisions based on what the agent needs for its next action. Further details are in Appendix A.3.

3 Experimental Setup

We evaluate VTR in two domains: computer-use tasks and video games.

Models. We apply VTR to Qwen3.5-2B [Qwen Team, 2026]. We train a single model on both desktop tasks and video games, and evaluate it in both domains. The router is a three-layer MLP with 23.3M parameters, about 1% of the size of the language model, which reads each image region across the current and $K = 10$ previous observations. We place an anchor every $B = 10$ observations, and set the target $r^\star = 0.1$ (§2.3). Further details are in Appendix A.

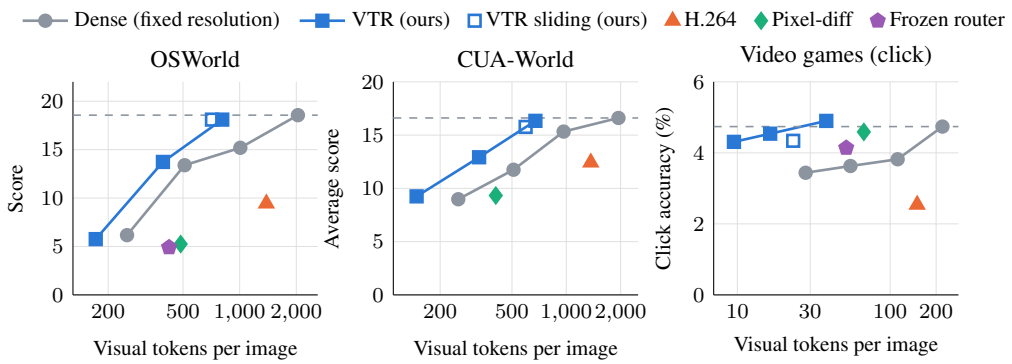


Figure 4: **VTR matches full-resolution performance with far fewer visual tokens.** Score on OSWorld (left), average score on CUA-World (middle), and click accuracy in video games (right), against visual tokens per image. The dashed line marks the dense model at full resolution.

Desktop tasks. We train on computer-use trajectories from CUA-World train, in which each screenshot is at a resolution of 1920 × 1080 giving 2,040 visual tokens. We evaluate the agent on OSWorld [Xie et al., 2024], which contains tasks in common desktop applications such as web browsers and office software, and on CUA-World [Aggarwal et al., 2026b], which contains tasks across 200 professional software applications.

Video games. We train on gameplay recordings at 30 frames per second from the `markov-ai/gaming-500-hours` dataset, with each frame resized to 640 × 360, which gives 220 visual tokens. We evaluate how accurately the agent predicts the next action, such as a click or a key press, on held-out recordings from 36 games.

Baselines. We compare VTR with the following ways to reduce visual tokens, all using the same model and training data:

1. **Lower resolution:** the dense model, which processes every visual token, with each image resized to 50%, 25%, or 12.5% of its area.
2. **Pixel-diff:** keeps only the regions whose pixels changed from the previous observation, following methods that drop tokens repeated across video frames [Choudhury et al., 2024, Bagrov et al., 2025] and screenshots [Abaskohi et al., 2026].
3. **H.264:** keeps only the regions that the H.264 video codec [Wiegand et al., 2003] encodes as moved or changed, following methods that select visual tokens from the motion vectors and residuals of video codecs [Tang et al., 2026, Sarkar et al., 2026].
4. **Frozen router:** the initial router of VTR, kept fixed while training the model, which shows the effect of training the router jointly with the model.

As in VTR, both rules keep the anchors whole. Each baseline is trained with same experimental setup and common hyperparameters, as VTR. We also evaluate VTR with the sliding router (§2.2), and with images resized to half or a quarter of their area.

4 Results

4.1 Main Results

VTR matches full-resolution performance with far fewer visual tokens. Figure 4 compares VTR with the dense model, which processes every visual token, across numbers of tokens per image. On OSWorld and CUA-World, VTR matches the dense model at full resolution while using 2.5× and 2.9× fewer visual tokens, and in video games, it uses 5.8× fewer. Interestingly, the same router keeps about 40% of the tokens on OSWorld but only 17% in video games. Since we never specify how many tokens to keep (§2.3), the router learns this difference on its own, likely because consecutive game frames are only 1/30 of a second apart, while desktop screenshots are taken after

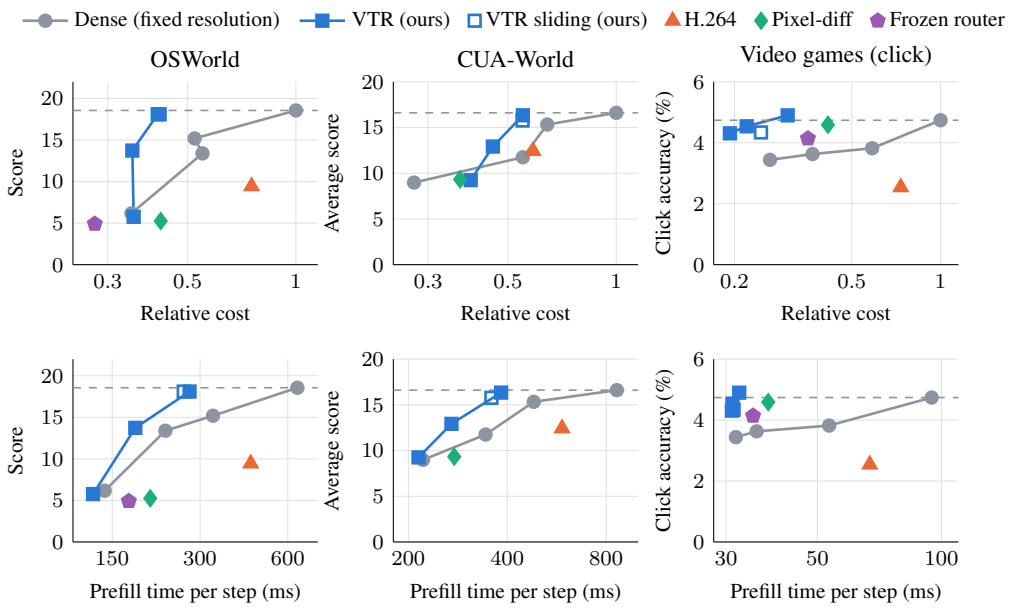


Figure 5: **VTR makes inference cheaper and faster.** Score on OSWorld (left), average score on CUA-World (middle), and click accuracy in video games (right), against inference cost relative to the dense model at full resolution (top) and prefill time per step (bottom). The dashed line marks the dense model at full resolution.

each action. These results show that most visual tokens in an agent's history can be removed without degrading its actions, and that the router adapts how many it removes to the input.

VTR outperforms other token reduction approaches. Figure 4 also compares VTR with two other ways to reduce visual tokens: lowering the image resolution, and hand-designed rules that keep the regions that changed between observations, namely pixel-diff and H.264. At every number of tokens where both are measured, VTR achieves higher performance than the dense model at a lower resolution. For instance, on OSWorld, VTR scores 2.9 points higher than the dense model at half the image area, while using fewer tokens. Lowering the resolution removes detail from every region of the screen, including the small regions that actions depend on, like a cursor or a spreadsheet cell, whereas VTR keeps the regions it selects at full resolution. VTR also outperforms both hand-designed rules on every benchmark. On OSWorld, for instance, VTR scores 18.1, compared with 9.4 for H.264 and 5.3 for pixel-diff. Unlike these rules, which decide what to keep from the pixels alone, VTR learns what the agent needs for its next action in each domain. Moreover, VTR is complementary to lowering the resolution: combined with a quarter of the image area, it reduces visual tokens by 23× in video games, with only a small drop in click accuracy. These results show that learning which regions to keep preserves more of what the agent needs than lowering the resolution or applying a fixed rule.

Learning which tokens to keep jointly with the model is crucial. To isolate the effect of joint training, we compare VTR with a frozen router, which starts from the same initial router (§2.3) but keeps it fixed while training the language model. On OSWorld, VTR scores 18.1, compared with only 4.9 for the frozen router, which performs similarly to pixel-diff. This is expected, since the initial router is trained on a similar rule, keeping regions that changed and belong to an object, and training the language model alone cannot recover the regions that this router drops. In video games, VTR also achieves higher click accuracy than the frozen router, although the gap is smaller. These results show that the advantage of VTR comes from training the router jointly with the model, rather than from the initial rule.



Figure 7: **VTR keeps only the regions the agent needs.** The previous observation (top) and the current observation with the regions that VTR drops dimmed (bottom), in three video games and a desktop application.

4.2 Analysis

VTR makes inference cheaper and faster. Since VTR removes most visual tokens, each step of the agent also costs less and takes less time. Figure 5 shows the performance of each method against its inference cost and its prefill time, the time the model takes to process its input before generating an action. VTR reduces the cost by 2.4× on OSWorld, 1.8× on CUA-World, and 3.3× in video games, and speeds up prefill by 2.3×, 2.2×, and 2.9×. These gains are smaller than the reduction in visual tokens, since VTR removes only visual tokens, while the instruction, the previous actions, and the generated action stay the same. Moreover, VTR is both cheaper and faster than the dense model at half the image area in all three domains, while achieving higher performance. These results show that the token savings of VTR translate into lower cost and faster inference, while matching full-resolution performance.

VTR adapts how many visual tokens it keeps to each domain and game. Since we never fix the number of kept tokens (§2.3), the router decides how many tokens each input needs. To see how this number varies, we group the video games by how much of the screen changes between consecutive frames, from games like Candy Crush, where little of the screen moves, to games like Counter-Strike 2, where most of it does. Figure 6 shows the percentage of visual tokens each method keeps. VTR keeps 2.7% of the tokens in games with low motion, 5.9% in games with high motion, and 26.3% on desktop tasks. Interestingly, as more of a game screen changes, VTR keeps only a few more tokens, while pixel-diff and H.264 keep many more. This is likely because many regions that change, such as the background moving as the camera turns, do not affect the agent's next action. On desktop tasks, in contrast, VTR keeps more tokens than pixel-diff, likely because the agent often needs regions that did not change, such as text it has to read. These results show that VTR decides how many visual tokens to keep based on what each input needs, rather than on how much of the screen changes.

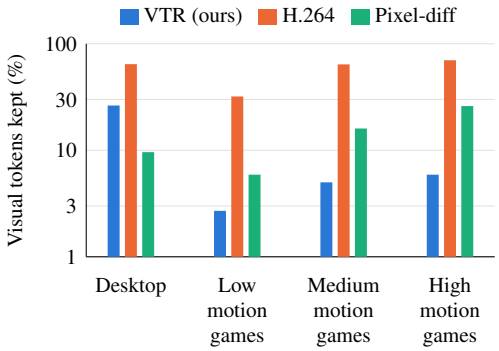


Figure 6: **VTR adapts how many visual tokens it keeps to each domain and game.** Percentage of visual tokens kept by each method on desktop tasks and on video games grouped by how much of the screen changes between consecutive frames.

VTR keeps the regions the agent needs for its next action. To see which regions the router keeps, Figure 7 shows the previous observation and the regions that VTR keeps in the current one. In Dota 2, a fight takes place on an otherwise static map, and VTR keeps 23 of the 220 regions, mostly the fighting units and their health bars. In Candy Crush, the player moves a single candy, and VTR keeps only the region of that candy. In Minecraft, VTR keeps the opponent and the health

counter that dropped from 13 to 11, and drops the field around them. Interestingly, in Gpredict, VTR keeps the region where a tooltip disappeared, so it also treats the removal of an element as information, while it drops the satellite map behind the dialog. VTR also keeps a few isolated regions in static areas, likely because each gate is sampled from its keep probability. Overall, rather than just detecting motion, the router appears to keep the changes that the model cannot infer from earlier observations, such as a lower health counter or a closed tooltip, and to drop those it can, such as the shifted background when the camera moves in GTA V (Appendix B.1). Further examples are in Appendix B.1.

5 Related Work

Visual token reduction. Many methods reduce the visual tokens of vision-language models without training, by keeping the tokens that receive the most attention [Chen et al., 2024, Yang et al., 2025, Zhang et al., 2025b], merging similar tokens [Bolya et al., 2023, Shang et al., 2025], or dropping tokens that repeat across video frames, detected from pixels or from video codecs [Shen et al., 2025, Bagrov et al., 2025, Choudhury et al., 2024, Tang et al., 2026, Sarkar et al., 2026]. Other methods train a module that predicts which tokens to keep [Rao et al., 2021, Huang et al., 2025, Ye et al., 2025, Wang et al., 2026a]. However, all of these methods set the number of kept tokens in advance, as a fixed ratio, a fixed threshold, or a target token count. In contrast, VTR trains its router jointly with the language model and keeps as many tokens as the model needs to preserve its accuracy.

Efficient computer-use agents. Computer-use agents take far longer than humans to complete tasks [Abhyankar et al., 2026, Aggarwal et al., 2026a], and latency limits them in real-time domains such as video games [Zhang et al., 2025a, Aggarwal and Welleck, 2026]. To reduce the cost of screenshots, agents commonly keep only the last few screenshots in context [Qin et al., 2025, Wang et al., 2025a, Anthropic, 2026], and methods designed for GUI agents skip, prune, or compress visual tokens [Lin et al., 2025, Xu et al., 2026, Huang et al., 2026, Chen et al., 2025, Wang et al., 2026b, Abaskohi et al., 2026]. However, these methods either discard older screenshots entirely, or decide what to drop with fixed rules or a selector trained separately from the model that predicts actions. In contrast, VTR learns which tokens to keep together with the model, based on what the agent needs for its next action.

Learned routing and constrained optimization. Mixture-of-experts and Mixture-of-Depths models learn to route tokens through parts of the network [Shazeer et al., 2017, Fedus et al., 2022, Zhou et al., 2022, Raposo et al., 2024], and adaptive computation methods let the amount of computation vary across inputs [Graves, 2016, Wang et al., 2025b, Schuster et al., 2022]. Separately, constrained optimization adapts a Lagrange multiplier during training to meet a target, for example in VAEs, reinforcement learning, and pruning [Rezende and Viola, 2018, Tessler et al., 2019, Wang et al., 2020, Gallego-Posada et al., 2022]. However, these methods fix the amount of computation, set a target for it, or calibrate it only after training, rather than constraining the model to keep its accuracy during training. In contrast, VTR minimizes the number of kept tokens subject to the model not losing accuracy, and enforces this constraint during training by dual ascent. Appendix C provides an extended discussion of related work.

6 Conclusion

In this work, we introduced Visual Token Routing (VTR), a joint training framework that enables computer-use agents to process only the visual tokens they need. By training a lightweight spatio-temporal router together with the vision-language model to keep as few visual tokens as possible without losing accuracy, VTR addresses a key computational bottleneck of computer-use agents, which arises from representing each screenshot with thousands of visual tokens. VTR matches the performance of the full-resolution model on OSWorld, CUA-World, and video game action prediction with 2.5-5.8$\times$ fewer visual tokens, reducing inference cost by up to 3.3$\times$ and speeding up prefill by up to 2.9$\times$, and outperforms lowering the image resolution and other baselines for dropping tokens. We further show that learning which tokens to keep jointly with the model is crucial. Our analysis also shows that the router learns on its own how many tokens to keep for each input and domain, keeping only regions that matter for the agent's next action. By learning what

an agent needs to see, VTR makes it possible to use computer-use agents in long-horizon tasks and real-time domains at a fraction of the computational cost.

Acknowledgements

We thank the CMU Foundation and Language Model (FLAME) Center for providing computational resources. We thank Jing Yu Koh for their feedback and helpful discussions. Pranjal Aggarwal was supported by SoftBank Group-Arm Fellowship during the project. This work is partially supported by the National Science Foundation under Grants DMS-2502281.

References

Amirhossein Abaskohi, Yuhang He, Peter West, Giuseppe Carenini, Pranit Chawla, and Vibhav Vineet. ReVision: Scaling computer-use agents via temporal visual redundancy reduction. *arXiv preprint arXiv:2605.11212*, 2026.

Reyna Abhyankar, Qi Qi, and Yiying Zhang. OSWorld-Human: Benchmarking the efficiency of computer-use agents. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 8, 2026.

Pranjal Aggarwal and Sean Welleck. Gym-Anything-RT: Scaling real-time environments for vision-language agents. *arXiv preprint arXiv:2610.37567*, 2026.

Pranjal Aggarwal, Lawrence Keunho Jang, Sean Welleck, Daniel Fried, Ruslan Salakhutdinov, and Jing Yu Koh. cua-speedrun: Standardized benchmarking of the speed of computer-use agents. *arXiv preprint arXiv:2609.40284*, 2026a.

Pranjal Aggarwal, Graham Neubig, and Sean Welleck. Gym-Anything: Turn any software into an agent environment. *arXiv preprint arXiv:2604.06126*, 2026b.

Eitan Altman. *Constrained Markov Decision Processes*. Chapman and Hall/CRC, 1999.

Anthropic. Computer use tool. `https://platform.claude.com/docs/en/agents-and-tools/tool-use/computer-use-tool`, 2026. Claude API documentation, accessed November 2026.

Natan Bagrov, Eugene Khvedchenia, Borys Tymchenko, Shay Aharon, Lior Kadoch, Tomer Keren, Ofri Masad, Yonatan Geifman, Ran Zilberstein, Tuomas Rintamaki, Matthieu Le, and Andrew Tao. Efficient video sampling: Pruning temporally redundant tokens for faster VLM inference. *arXiv preprint arXiv:2510.14624*, 2025.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your ViT but faster. In *International Conference on Learning Representations (ICLR)*, 2023.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Gongwei Chen, Xurui Zhou, Rui Shao, Yibo Lyu, Kaiwen Zhou, Shuai Wang, Wentao Li, Yinchuan Li, Zhongang Qi, and Liqiang Nie. Less is more: Empowering GUI agent with context-aware simplification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5901–5911, 2025.

Liang Chen, Haozhe Zhao, Tianyu Liu, Shuai Bai, Junyang Lin, Chang Zhou, and Baobao Chang. An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models. In *European Conference on Computer Vision (ECCV)*, 2024.

Rohan Choudhury, Guanglei Zhu, Sihan Liu, Koichiro Niinuma, Kris M. Kitani, and László A. Jeni. Don't look twice: Faster video transformers with run-length tokenization. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 28127–28149, 2024.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Jose Gallego-Posada, Juan Ramirez, Akram Erraqabi, Yoshua Bengio, and Simon Lacoste-Julien. Controlled sparsity via constrained optimization or: How I learned to stop tuning penalties and love constraints. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 1253–1266, 2022.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Yongxin Guo, Zhenglin Cheng, Xiaoying Tang, Zhaopeng Tu, and Tao Lin. Dynamic mixture of experts: An auto-tuning approach for efficient transformer models. In *International Conference on Learning Representations (ICLR)*, 2025.

Kung-Hsiang Huang, Haoyi Qiu, Yutong Dai, Caiming Xiong, and Chien-Sheng Wu. GUI-KV: Efficient GUI agents via KV cache with spatio-temporal awareness. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856.

Wenxuan Huang, Zijie Zhai, Yunhang Shen, Shaosheng Cao, Fei Zhao, Xiangfeng Xu, Zheyu Ye, and Shaohui Lin. Dynamic-LLaVA: Efficient multimodal large language models via dynamic vision-language context sparsification. In *International Conference on Learning Representations (ICLR)*, 2025.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-softmax. In *International Conference on Learning Representations (ICLR)*, 2017.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2023.

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

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations (ICLR)*, 2017.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. UI-TARS: Pioneering automated GUI interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL `https://qwen.ai/blog?id=qwen3.5`.

Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. DynamicViT: Efficient vision transformers with dynamic token sparsification. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 13937–13949, 2021.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-Depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

Danilo Jimenez Rezende and Fabio Viola. Taming VAEs. *arXiv preprint arXiv:1810.00597*, 2018.

Sayan Deb Sarkar, Rémi Pautrat, Ondrej Miksik, Marc Pollefeys, Iro Armeni, Mahdi Rad, and Mihai Dusmanu. CoPE-VideoLM: Leveraging codec primitives for efficient video language modeling. *arXiv preprint arXiv:2602.13191*, 2026.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 17456–17472, 2022.

Yuzhang Shang, Mu Cai, Bingxin Xu, Yong Jae Lee, and Yan Yan. LLaVA-PruMerge: Adaptive token reduction for efficient large multimodal models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 22857–22867, 2025.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017.

Xiaoqian Shen, Yunyang Xiong, Changsheng Zhao, Lemeng Wu, Jun Chen, Chenchen Zhu, Zechun Liu, Fanyi Xiao, Balakrishnan Varadarajan, Florian Bordes, Zhuang Liu, Hu Xu, Hyunwoo J. Kim, Bilge Soran, Raghuraman Krishnamoorthi, Mohamed Elhoseiny, and Vikas Chandra. LongVU: Spatiotemporal adaptive compression for long video-language understanding. In *International Conference on Machine Learning (ICML)*, 2025.

Adam Stooke, Joshua Achiam, and Pieter Abbeel. Responsive safety in reinforcement learning by PID Lagrangian methods. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 9133–9143. PMLR, 2020.

Feilong Tang, Xiang An, Yunyao Yan, Yin Xie, Bin Qin, Kaicheng Yang, Yifei Shen, Yuanhan Zhang, Chunyuan Li, Shikun Feng, Changrui Chen, Huajie Tan, Ming Hu, Manyuan Zhang, Bo Li, Ziyong Feng, Ziwei Liu, Zongyuan Ge, and Jiankang Deng. OneVision-Encoder: Codec-aligned sparsity as a foundational principle for multimodal intelligence. *arXiv preprint arXiv:2602.08683*, 2026.

Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. Reward constrained policy optimization. In *International Conference on Learning Representations (ICLR)*, 2019.

Shida Wang, YongXiang Hua, Zhou Tao, Haoyu Cao, and Linli Xu. Dynamic token compression for efficient video understanding through reinforcement learning. *arXiv preprint arXiv:2603.26365*, 2026a.

Shuo Wang, Sifan Zhou, Xiaobo Wang, Tianyu Fu, Chenxu Zhao, Anyang Su, Hanning Wang, Ping Jiang, and Minghui Wu. STP-GUI: Spatially-aware token pruning for efficient web-GUI agents. *arXiv preprint arXiv:2610.20390*, 2026b.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, Yeqiao Fu, Dongchan Shin, Martin Shin, Jiarui Hu, Yuyan Wang, Jixuan Chen, Yuxiao Ye, Danyang Zhang, Yipu Wang, Heng Wang, Diyi Yang, Victor Zhong, Y. Charles, Zhilin Yang, and Tao Yu. OpenCUA: Open foundations for computer-use agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, pages 139756–139806, 2025a.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. Structured pruning of large language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6151–6162. Association for Computational Linguistics, 2020.

Ziteng Wang, Jun Zhu, and Jianfei Chen. ReMoE: Fully differentiable mixture-of-experts with ReLU routing. In *International Conference on Learning Representations (ICLR)*, 2025b.

Thomas Wiegand, Gary J. Sullivan, Gisle Bjøntegaard, and Ajay Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2003.

Shiwei Wu, Joya Chen, Kevin Qinghong Lin, Qimeng Wang, Yan Gao, Qianli Xu, Tong Xu, Yao Hu, Enhong Chen, and Mike Zheng Shou. VideoLLM-MoD: Efficient video-language streaming with mixture-of-depths vision computation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 109922–109947, 2024.

Mengzhou Xia, Zexuan Zhong, and Danqi Chen. Structured pruning learns compact and accurate models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1513–1528. Association for Computational Linguistics, 2022.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *NeurIPS*, 2024.

Zhou Xu, Bowen Zhou, Qi Wang, Shuwen Feng, and Jingyu Xiao. Spatio-temporal token pruning for efficient high-resolution GUI agents. *arXiv preprint arXiv:2602.23235*, 2026.

Senqiao Yang, Yukang Chen, Zhuotao Tian, Chengyao Wang, Jingyao Li, Bei Yu, and Jiaya Jia. VisionZip: Longer is better but not necessary in vision language models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Xubing Ye, Yukang Gan, Yixiao Ge, Xiao-Ping Zhang, and Yansong Tang. ATP-LLaVA: Adaptive token pruning for large vision language models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Hongxu Yin, Arash Vahdat, Jose M. Alvarez, Arun Mallya, Jan Kautz, and Pavlo Molchanov. A-ViT: Adaptive tokens for efficient vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10809–10818, 2022.

Alex L. Zhang, Thomas L. Griffiths, Karthik R. Narasimhan, and Ofir Press. VideoGameBench: Can vision-language models complete popular video games? *arXiv preprint arXiv:2505.18134*, 2025a.

Yuan Zhang, Chun-Kai Fan, Junpeng Ma, Wenzhao Zheng, Tao Huang, Kuan Cheng, Denis A. Gudovskiy, Tomoyuki Okuno, Yohei Nakata, Kurt Keutzer, and Shanghang Zhang. SparseVLM: Visual token sparsification for efficient vision-language model inference. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pages 74840–74857. PMLR, 2025b.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M. Dai, Zhifeng Chen, Quoc V. Le, and James Laudon. Mixture-of-experts with expert choice routing. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 7103–7114, 2022.

A Experimental Details

A.1 Training Data

The training data contains 4,265 computer-use examples from the CUA-World training split [Aggarwal et al., 2026b] and 29,401 video game examples from the `markov-ai/gaming-500-hours` dataset, and we sample computer-use and video game examples at a ratio of 1:3. Each example contains up to 20 screenshots for computer use and 40 frames for video games, with older steps given as text, and we compute the cross-entropy loss on every token of the target action.

A.2 Router Architecture

The router's input concatenates the visual tokens of $K + 1 = 11$ observations, which gives $11 \times 2{,}048 = 22{,}528$ dimensions. The MLP has layers of size 22,528, 1,024, 256, and 1, with GELU activations and a dropout of 0.1, for a total of 23.3M parameters. Requests with a single observation are kept in full.

A.3 Initialization

For the initialization labels (§2.3), a region counts as changed if its mean absolute pixel change from the previous observation is at least 10 out of 255. We train on windows of 11 observations and supervise only the last one, using 47,489 computer-use windows and 29,709 video game windows. We train the router with a binary cross-entropy loss for 558 steps, with a batch size of 1,024 and AdamW with a learning rate of 1.5×10^{-3}.

A.4 Joint Training

Table 1 lists the hyperparameters for joint training. The full and blind models each have their own AdamW optimizer, and the visual tokens of each example are computed once and shared by all three models. For the adaptive weight, we use a 10 times larger step size when $\bar{r}$ is below $r^\star$, so that λ decreases quickly and the router resumes dropping tokens.

Table 1: Hyperparameters for joint training.

Hyperparameter	**Value**
Base model	Qwen3.5-2B
Vision encoder	Frozen
Precision	BF16
Context length	65,536 tokens
Batch size	32
Hardware	8 H100 GPUs
Training steps	2,000
Optimizer	AdamW
Model learning rate	10^{-5}, constant
Router learning rate	5×10^{-5}
Weight decay (model / router)	0.01 / 10^{-4}
Gradient clipping (model)	1.0
Router parameters	23.3M
History length K	10
Anchor interval B	10
Degradation target $r^\star$	0.1
Step size α	0.2
Initial weight λ_0	0.5

A.5 Baselines

The dense, pixel-diff, and H.264 baselines are evaluated after 1,000 training steps, and the frozen router after 2,000.

Table 2: Results on OSWorld and CUA-World for all methods. Tokens are visual tokens per image. Cost and prefill time are reductions relative to the dense model at full resolution.

	OSWorld				CUA-World			
Method	**Tokens**	**Score**	**Cost**	**Prefill**	**Tokens**	**Avg. score**	**Cost**	**Prefill**
Dense, 100% area	2040.0	18.56	1.0×	1.0×	1946.2	16.61	1.0×	1.0×
Dense, 50% area	1008.0	15.19	1.9×	1.9×	966.5	15.33	1.6×	1.8×
Dense, 25% area	510.0	13.39	1.8×	2.8×	510.0	11.75	1.8×	2.5×
Dense, 12.5% area	252.0	6.17	2.9×	4.6×	252.0	8.98	3.6×	3.9×
H.264	1386.5	9.44	1.3×	1.4×	1367.0	12.43	1.7×	1.5×
Pixel-diff	485.9	5.25	2.4×	3.2×	406.8	9.33	2.7×	3.1×
Frozen router	421.5	4.91	3.6×	3.8×	N/A	N/A	N/A	N/A
VTR (ours)	807.9	18.10	2.4×	2.3×	674.3	16.34	1.8×	2.2×
VTR sliding (ours)	712.6	18.10	2.4×	2.5×	595.2	15.77	1.8×	2.4×
VTR, 1/2 area (ours)	391.0	13.73	2.8×	3.6×	328.8	12.92	2.2×	3.2×
VTR, 1/4 area (ours)	171.8	5.76	2.8×	5.0×	148.1	9.25	2.5×	4.0×

Dense. Images are resized to 100%, 50%, 25%, and 12.5% of their area, which gives 2,040, 1,008, 510, and 252 visual tokens per screenshot, and 220, 112, 55, and 28 per game frame.

Pixel-diff. Pixel-diff uses the same threshold as the initialization labels (Appendix A.3).

H.264. We encode the observations with libx264 at a quantization parameter of 0, with a keyframe every 10 frames to match the anchors. A region counts as changed if it has a nonzero motion vector or a nonzero residual, and keyframes are kept in full.

Frozen router. The model is trained with the cross-entropy loss alone, without the reference models or the adaptive weight.

VTR at lower resolution. Observations are downscaled by a linear factor of $1/\sqrt{2}$ or $1/2$ before encoding, with the same trained router and anchors.

A.6 Evaluation

OSWorld. We evaluate on the 295 of the 369 tasks that do not require internet access, with up to 100 steps per task, and average the results over three seeds.

CUA-World. We evaluate on 512 tasks, with up to 200 steps per task, and report the average score from its checklist verifier with Gemini 3 Flash.

Video games. The held-out data contains about 90,000 actions from 36 games, consisting of 30,000 clicks, 26,880 mouse moves, 3,120 drags, 30,000 key presses, and 113 scrolls. A click counts as correct if the predicted action is a click with the same button and phase, press or release, as the target.

Cost and prefill time. We compute the inference cost per task at prices of \$0.017, \$0.015 and \$0.025 per million input, cached input and output tokens, with prompt caching, and the prefill time as the time to prefill one request of the mean input length in vLLM on one NVIDIA L40S GPU.

B Additional Results

Table 3 tests whether VTR performs differently from the dense model at full resolution on OSWorld and CUA-World. On both benchmarks, the difference is small and not statistically significant.

Table 3: Difference between VTR and the dense model at full resolution, with a paired t-test over the tasks graded for both methods.

Benchmark	Metric	Difference	95% CI	p-value
OSWorld	Score	-0.46	$[-2.88,\ 1.94]$	0.71
OSWorld	Pass rate (%)	-0.34	$[-2.71,\ 2.03]$	0.77
CUA-World	Average score	-0.52	$[-2.78,\ 1.74]$	0.66
CUA-World	Pass rate (%)	$+0.20$	N/A	0.87

Table 4: Action prediction accuracy (%) in video games for all methods, by action type. Reduction is relative to the dense model at full resolution.

Method	Tokens	Reduction	Click	Move	Drag	Keyboard
Dense, 100% area	220.0	1.0×	4.74	23.33	14.90	4.98
Dense, 50% area	112.0	2.0×	3.82	22.28	16.28	5.23
Dense, 25% area	55.0	4.0×	3.63	22.91	15.96	4.76
Dense, 12.5% area	28.0	7.9×	3.44	23.39	16.15	5.04
H.264	150.3	1.5×	2.54	22.68	15.13	5.42
Pixel-diff	67.3	3.3×	4.59	24.48	15.58	5.86
Frozen router	51.6	4.3×	4.14	28.79	19.84	5.92
VTR (ours)	38.2	5.8×	4.90	24.94	16.86	5.37
VTR sliding (ours)	23.2	9.5×	4.34	25.41	17.05	5.41
VTR, 1/2 area (ours)	16.4	13.4×	4.54	24.19	16.38	4.84
VTR, 1/4 area (ours)	9.5	23.2×	4.31	23.90	16.19	4.76

B.1 Qualitative Examples

Figure 8 shows further examples of the regions that VTR keeps, in video games and desktop applications.

C Extended Related Work

This appendix expands on the related work in §5.

Visual token reduction. Vision-language models represent each image with hundreds to thousands of visual tokens, and a large body of work reduces this number to speed up inference. Training-free methods score tokens by attention, either inside the language model, as FastV [Chen et al., 2024] does after its second layer, or in the vision encoder, as VisionZip [Yang et al., 2025] does, while SparseVLM [Zhang et al., 2025b] scores visual tokens by the attention they receive from the relevant text tokens. Token merging combines similar tokens instead of dropping them, first in vision transformers [Bolya et al., 2023] and then in vision-language models, where LLaVA-PruMerge [Shang et al., 2025] also chooses the number of kept tokens for each image with an outlier rule on attention scores. For video, methods exploit redundancy across frames: LongVU [Shen et al., 2025] drops near-duplicate frames and reduces tokens based on their similarity across frames, EVS [Bagrov et al., 2025] removes patches that stay the same across consecutive frames while keeping the positions of the remaining tokens, and run-length tokenization [Choudhury et al., 2024] replaces patches that repeat over time with a single token. Codec-based methods instead read the motion vectors and residuals that video codecs already compute: OneVision-Encoder [Tang et al., 2026] selects the patches with the strongest motion and residuals across frames, up to a fixed token budget, and trains its vision encoder on these sparse inputs, while CoPE-VideoLM [Sarkar et al., 2026] encodes only keyframes with the image encoder and represents the other frames with a few tokens built from motion vectors and residuals. Learned methods train a module that predicts which tokens to keep together with the model, starting with DynamicViT [Rao et al., 2021] for image classification and extending to vision-language models with Dynamic-LLaVA [Huang et al., 2025] and ATP-LLaVA [Ye et al., 2025], while VideoLLM-MoD [Wu et al., 2024] learns which video tokens skip the computation of each layer. However, in all of these methods, the number of kept tokens is set by hand: training-free methods use a fixed ratio or threshold, codec-based methods use a fixed token budget or

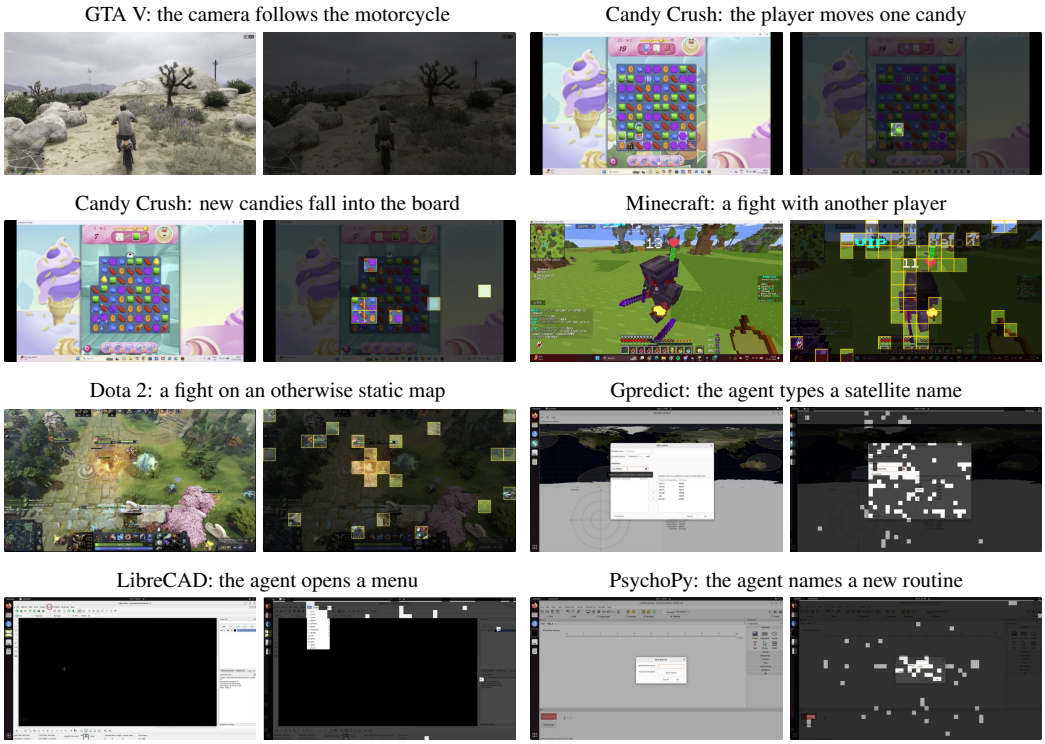


Figure 8: **Further examples of the regions that VTR keeps.** For each example, the previous observation (left) and the current observation with the regions that VTR drops dimmed (right), in video games and desktop applications.

keyframe density, DynamicViT and Dynamic-LLaVA add a loss that pulls the kept fraction toward a pre-defined keep rate, ATP-LLaVA pulls the average number of kept tokens toward a target count, and VideoLLM-MoD skips a fixed fraction of tokens. Moreover, apart from the video methods, they decide for each image independently. SCORE [Wang et al., 2026a] comes closest to our work. A small MLP reads each video token together with its change since the previous frame and outputs a keep probability, and the MLP is trained with reinforcement learning on sampled keep masks. However, SCORE keeps the vision encoder and the language model frozen, looks only one frame back, and keeps a fixed fraction of tokens at inference by selecting the highest-scoring ones. In contrast, the router in VTR reads each image region across several previous screenshots and is trained jointly with the language model, so the model learns to act from the kept tokens, and the number of kept tokens is not set in advance but chosen so that the model does not lose accuracy.

Efficient computer-use agents. As computer-use agents improve on benchmarks such as OSWorld [Xie et al., 2024], recent work has turned to their speed and cost. OSWorld-Human [Abhyankar et al., 2026] finds that agents take tens of minutes on tasks that humans complete in a few minutes, with later steps taking up to three times longer than early ones, and cua-speedrun [Aggarwal et al., 2026a] standardizes the infrastructure for measuring the speed and cost of agents across benchmarks. In real-time domains, VideoGameBench [Zhang et al., 2025a] identifies inference latency as a major limitation of frontier models in video games, and in Gym-Anything-RT [Aggarwal and Welleck, 2026], the pass rate of a frontier agent rises from 9.0% to 75.0% when the environment is paused while the agent decides. To limit the cost of screenshots, agents commonly keep only the last few screenshots in context and represent earlier steps as text [Qin et al., 2025, Wang et al., 2025a], and Anthropic recommends reducing the resolution of screenshots and keeping only the most recent ones [Anthropic, 2026]. Methods designed for GUI agents reduce tokens more selectively. ShowUI [Lin et al., 2025] groups patches of similar color into connected components and skips part of the patches in each component, GUIPruner [Xu et al., 2026] lowers the resolution of older screenshots and prunes the current one with rules that preserve its layout, GUI-KV [Huang et al., 2026] compresses the key-value cache by scoring how redundant older screenshots are given

the current one, and SimpAgent [Chen et al., 2025] trains the model to drop all history screenshots after a fixed layer. STP-GUI [Wang et al., 2026b] learns a token selector with reinforcement learning, but keeps the model frozen and keeps a fixed fraction of tokens. However, none of these methods learns which tokens to keep together with the model that predicts actions. ReVision [Abaskohi et al., 2026] comes closest to our work. A three-layer MLP compares each patch with the same patch in the previous screenshot and drops unchanged patches, the first screenshot in the window is kept in full, and kept patches retain their original positions. However, ReVision trains its MLP on labels obtained by matching regions from a UI parser across screenshots, and only then fine-tunes the model on the filtered trajectories, so the MLP detects unchanged content rather than learning what the agent needs for its next action. In contrast, VTR starts its router from similar labels, which combine changed regions with segmented objects (§2.3), and then trains the router jointly with the model to preserve action prediction, on desktop tasks and video games.

Learned routing and constrained optimization. Conditional computation trains a router to decide which parts of a network process each input. Sparsely gated mixture-of-experts models route each token to its top-scoring experts [Shazeer et al., 2017, Fedus et al., 2022], expert choice routing lets each expert pick a fixed number of tokens [Zhou et al., 2022], and Mixture-of-Depths [Raposo et al., 2024] applies the same idea across layers, sending a fixed number of tokens through each block while the rest skip it. In these models, the router's weight scales the output, which gives the router a gradient despite the discrete choice, while other methods train discrete gates with the straight-through estimator [Bengio et al., 2013] or with continuous relaxations such as Gumbel-softmax [Jang et al., 2017, Maddison et al., 2017]. Adaptive computation methods let the amount of computation vary across inputs, by learning when to stop computing under a penalty on the amount of computation [Graves, 2016, Yin et al., 2022], or by activating a variable number of experts [Wang et al., 2025b, Guo et al., 2025], where ReMoE [Wang et al., 2025b] adapts the weight of its sparsity penalty to reach a target average sparsity. Separately, constrained optimization replaces hand-tuned loss weights with Lagrange multipliers that are adapted during training to meet a target, as in constrained VAEs [Rezende and Viola, 2018], constrained reinforcement learning [Altman, 1999, Tessler et al., 2019, Stooke et al., 2020], and pruning to a target sparsity [Wang et al., 2020, Xia et al., 2022, Gallego-Posada et al., 2022]. However, methods that reduce computation fix its amount, set a target for it, or weight it with a hand-tuned penalty, so they do not ensure that the model keeps its accuracy. CALM [Schuster et al., 2022] comes closest to our work, letting each token exit the language model early and calibrating the exit thresholds after training so that, with high probability, the outputs stay close to those of the full model. In contrast, VTR minimizes the number of kept tokens subject to the model not losing accuracy, measured against reference models with and without the full history, and enforces this constraint during training by dual ascent. Like mixture-of-experts models, VTR scales each kept token by its keep probability, but it removes dropped tokens from the input entirely, so they cost no computation in any layer.