CAN MULTI-MODAL LLMS PROVIDE LIVE STEP-BY-STEP TASK GUIDANCE?

Apratim Bhattacharyya[1]* **Bicheng Xu**[2]*† **Sanjay Haresh**[1] **Reza Pourreza**[1]
Litian Liu[1] **Sunny Panchal**[1] **Pulkit Madan**[1] **Leonid Sigal**[2] **Roland Memisevic**[1]
[1] Qualcomm AI Research‡ [2] University of British Columbia

ABSTRACT

Current state of the art multi-modal Large Language Models (LLM) have advanced conversational abilities. However, their effectiveness as coaches for learning everyday skills by providing live, interactive step-by-step guidance is still untested. Effective guidance requires not only delivering instructions but also detecting their successful execution, as well as identifying and alerting users to mistakes, all of which has to happen in real-time. To evaluate such capabilities, we introduce LIVECOOK, a new benchmark and dataset built upon CaptainCook4D, features densely annotated, timed instructions and feedback messages, specifically including mistake alerts precisely timestamped to their visual occurrence in the video. Extensive evaluation shows that current state of the art multi-modal LLMs struggle with providing live, interactive step-by-step guidance.

1 INTRODUCTION

Traditionally, learning everyday skills (e.g., cooking a new dish) means reading a recipe or watching a video, since domain experts like chefs are not available to teach us in person. Advances in multimodal large language models now allow AI systems to understand and respond to speech, audio, and visual inputs in real time. This creates an opportunity to leverage such models for live, step-by-step guidance by emulating domain experts.

However, this calls for the multi-modal LLMs to be able to react interactively to events in the video streams. Consider an example where a multi-modal LLM guides a user while making, *e.g.*, Bruschetta in Fig. 1. At the stage where the tomatoes are being sliced, an instruction with the desired thickness the tomatoes needs to be provided. Then the model needs to detect whether the user has sliced the tomatoes to the desired thickness, and in the case of a mistake, *e.g.*, the user adds pepper instead of sugar to the batter, the model needs to alert the user as soon as it observes the mistake.

Specifically, for live step-by-step task guidance, we identify three key abilities curial for multi-modal LLMs to possess , i) providing the appropriate instruction based on current task progress, ii) detecting if the instruction has been successfully accomplished by the user, iii) detecting any mistakes made by the user and alerting the user as soon as possible. To evaluate multi-modal LLMs on live step-by-step task guidance, we introduce the LIVECOOK dataset and benchmark, as currently available large-scale vision-language datasets and benchmarks [6, 12, 28] inadequately capture such interactive scenarios. These existing datasets and benchmarks largely consist of participants recording their daily activities or expert demonstrations, which are insufficient for effectively assessing multi-modal LLMs' ability to provide step-by-step instructions, as they lack the critical scenarios where users make mistakes or diverges from the plan. We describe the LIVECOOK Benchmark and Dataset in detail next.

LIVECOOK Benchmark and Dataset. Our benchmark and dataset uses the videos from the CaptainCook4D [26] dataset as it contains actions with mistakes. This allows us to create a setup akin to a "non-reactive simulation", where we task the multi-modal LLM to produce the right instruction and feedback at the appropriate time, but the subject is non-compliant. Such a setup still provides us

*Equal contribution.
†Work done while employed at Qualcomm AI Research.
‡Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

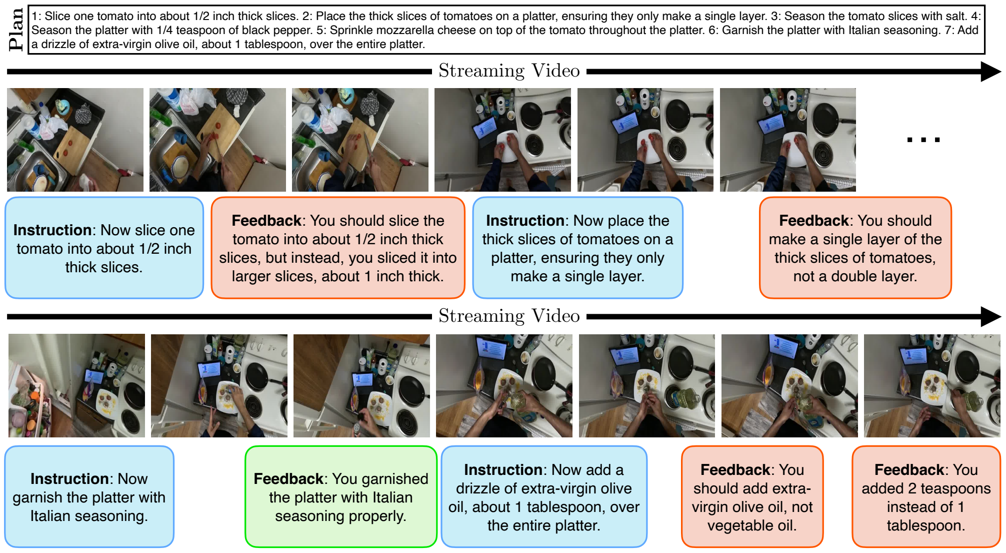

Plan
1: Slice one tomato into about 1/2 inch thick slices. 2: Place the thick slices of tomatoes on a platter, ensuring they only make a single layer. 3: Season the tomato slices with salt. 4: Season the platter with 1/4 teaspoon of black pepper. 5: Sprinkle mozzarella cheese on top of the tomato throughout the platter. 6: Garnish the platter with Italian seasoning. 7: Add a drizzle of extra-virgin olive oil, about 1 tablespoon, over the entire platter.
Streaming Video
Instruction: Now slice one tomato into about 1/2 inch thick slices.
Feedback: You should slice the tomato into about 1/2 inch thick slices, but instead, you sliced it into larger slices, about 1 inch thick.
Instruction: Now place the thick slices of tomatoes on a platter, ensuring they only make a single layer.
Feedback: You should make a single layer of the thick slices of tomatoes, not a double layer.
Streaming Video
Instruction: Now garnish the platter with Italian seasoning.
Feedback: You garnished the platter with Italian seasoning properly.
Instruction: Now add a drizzle of extra-virgin olive oil, about 1 tablespoon, over the entire platter.
Feedback: You should add extra-virgin olive oil, not vegetable oil.
Feedback: You added 2 teaspoons instead of 1 tablespoon.

Figure 1: An overview of the step-by-step task guidance scenario in our LIVECOOK benchmark and dataset, where the multi-modal LLM provides instructions and feedback that are sufficient to guide the user towards the goal, *e.g.*, making a tomato mozzarella salad (above).

with useful insight into ability of multi-modal LLMs to provide step-by-step task guidance, as fully reactive setups are not possible with offline datasets.

Annotations. The instructions and feedback are annotated using the following protocol, starting from a step-by-step plan. The first instruction of the plan occurs at the beginning of the video. If the user makes a mistake while completing the instruction, our benchmark and dataset contains a corresponding feedback just after it occurs. If an instruction is completed successfully, it is acknowledged (Fig. 1). Once all the steps are completed, we acknowledge the completion for the whole plan. We use the videos from the test set of CaptainCook4D as the LIVECOOK benchmark.

2 Baseline Methods for Step-by-Step Task Guidance

Multi-modal LLMs have seen amazing progress in recent years [1, 15, 17, 19, 24, 30, 35]. While they can take video as input, they only enable turn based interaction, *i.e.*, answering questions given the whole video. On the other hand, VideoLLM-online [3] and LiveCC [4] propose online narration frameworks for long videos. Because existing "turn-based" and narration models do not support live, step-by-step guidance, we next present a framework that adapts them for live task guidance. Finally, we also introduce a fine-tuned interactive model: LIVEMAMBA.

"Turn-based" multi-modal LLMs. To employ state of the art "turn-based" multi-modal LLMs such as [30] for live step-by-step task guidance, we need to employ a special online prompting strategy. This strategy involves prompting the model at regular intervals to detect both successful completions of instructions and mistakes. However prompting after every input frame is not computationally feasible. To balance accuracy and compute requirements, we prompt the models at an interval of 5 seconds. First, the model is asked if the user has completed the current instruction (obtained from the recipes in the LIVECOOK benchmark). If the model answers "yes", then we move on to the next instruction in the recipe. If not, then we ask the model to check for mistakes. In the offline (turn-based) evaluation scheme used in CaptainCook4D [26], multi-modal LLMs are prompted with sequence of questions per error category to detect mistakes. However, such a scheme is infeasible in case of streaming setups due to high computational costs. Therefore, to enable mistake detection in a single inference step, we design a prompt that concisely explains the possible mistakes that are possible while following a given instruction. The model can then use this information to recognize possible mistakes. In detail, for a given instruction we find similar instructions in the training set and use the Qwen-2.5-32B LLM [7] to summarize the possible mistakes. We provide details of the prompts in the appendix.

Table 1: Zero-shot evaluation on the LIVECOOK benchmark.

	Instruction	Mistake				
Method	IC-Acc↑	Prec.↑	Rec.↑	F1↑	BERT↑	ROUGE-L↑
LLaVA-NeXT [22]	1.4	0.00	0.00	0.00	0.000	0.000
Video-ChatGPT [24]	1.6	0.00	0.00	0.00	0.000	0.000
VideoChat2 [18]	1.6	0.00	0.00	0.00	0.000	0.000
Video-LLaVA [36, 20]	2.0	0.00	0.00	0.00	0.000	0.000
VideoLLaMA3-7B [34]	1.8	0.00	0.00	0.00	0.000	0.000
Videollm-online [3]	0.03	0.02	**0.98**	0.04	0.332	0.248
Qwen2-VL-7B [32]	6.3	0.02	0.69	**0.05**	0.377	0.256
Qwen2.5-VL-7B [10]	18.9	**0.18**	0.01	0.02	0.299	0.219
Gemini-2.5-Flash [29]	**23.1**	0.01	0.22	0.02	*0.410*	*0.342*

Table 2: Evaluation of fine-tuned models on LIVECOOK (†indicates models fine-tuned by us).

	Instruction	Mistake				
Method	IC-Acc↑	Prec.↑	Rec.↑	F1↑	BERT↑	ROUGE-L↑
Videollm-online† [3]	7.6	0.04	0.01	0.01	0.434	0.412
LIVEMAMBA (w/o-Aug)	7.8	0.05	0.01	0.01	0.605	0.542
LIVEMAMBA (Ours)	**31.5**	**0.17**	**0.10**	**0.13**	*0.651*	*0.561*

Narration models. On the other hand, streaming multi-modal LLMs such as Videollm-online [3] are targeted at providing online narrations of input video streams. Such models are therefore not directly applicable to our LIVECOOK benchmark. Thus, to evaluate such models, we first ask the model to generate online narrations for the whole video. We then feed these narrations in an online manner to a "helper" LLM (Phi-3-mini-4k-Instruct [9]) that given the narrations and the action instruction, predicts if the action is completed or not. If "yes", we move on to the next instruction. If "not", we ask the "helper" LLM to predict if the narrations suggest a mistake has been made and provide feedback for correction.

Fine-tuned interactive model. We also experiment using a fine-tuned interactive model: LIVE-MAMBA (architecture in the appendix). As LIVECOOK demands fine-grained object and action understanding (*e.g.*, adding the correct amount of salt, coating a cup vs. a bowl with oil), it uses the highly performant InternViT-300M-448px-V2_5 [5] vision encoder with a Q-Former adapter [16]. InternViT produces M tokens per frame, which the Q-Former compresses to K tokens via four cross-attention layers. These K visual tokens are interleaved with text tokens for plans, instructions, and feedback, and then passed to the Mamba-130M [13] language backbone. To time responses appropriately, following [3, 25], we employ two special tokens, `<vision>` (to request the next frame) and `<response>` (to emit instructions or feedback).

3 EXPERIMENTS

We use the following metrics to measure both the ability of the models to detect successfully completed instructions and to provide feedback when mistakes occur.

Instruction Completion Accuracy (IC-Acc). IC-Acc measures the proportion of instructions that the model correctly detects as successfully completed by the user. Specifically, this requires that the model provides the correct instruction, the user completes it, and the model identifies this completion. To mitigate temporal annotation noise, we consider a prediction correct if it falls within a small window centered on the ground-truth completion time. In practice, a 30-second window is sufficient: it typically spans the last $\sim$25% of the current step and the first $\sim$25% of the next step in the LIVECOOK benchmark, balancing robustness to noise with accuracy.

Mistake Detection Precision (Prec.), Recall (Rec.) and F1. To calculate mistake detection Precision, Recall, and F1 scores in our interactive streaming setup, we define: • *Mistake True Positive*: A mistake detected by the model within a small temporal window centered on the timestamp of a ground truth mistake. • *Mistake False Negative*: A ground truth mistake that the model fails to detect within this temporal window. • *Mistake False Positive*: A mistake detected by the model when no corresponding ground truth mistake occurs within the temporal window. • *Mistake True Negative*: An instruction correctly followed by the user (no ground truth mistake) where the model correctly detects no mistake. We use the same temporal window size as in the IC-Acc metric.

Mistake Feedback Fluency (BERT and ROUGE-L). To measure the fluency of the models in providing appropriate feedback, we use the ROUGE-L and BERT scores. We only consider the fluency of feedback provided in case true positive mistake detections. That is, when the feedback is provided within the temporal window of a ground truth feedback as described above. Importantly, these scores are only meaningful when comparing models with similar true positive detection rates, since differences in detection accuracy can confound fluency comparisons. This is because models with lower detection rates produce fewer feedback instances, which can skew the distribution of ROUGE and BERT scores and make fluency appear artificially higher or lower.

3.1 EVALUATION ON THE LIVECOOK BENCHMARK

We report the (zero-shot) evaluation results in Tab. 1 using state of the art multi-modal LLMs. Overall, Gemini-2.5-Flash [31] performs best. It can recognize 18.9% of instructions being successfully completed by the user. Models such as VideoLLaMA3-7B [34], Video-LLaVA [20, 36], VideoChat2 [18], Video-ChatGPT [24], LLaVA-NeXT [22] have trouble following instructions. Even when prompted to detect if the person has completed a given instruction these models tend to answer "yes" too early. This highlights a gap in understanding scenes as they unfold in a streaming setup. This is likely an artifact of the question-answer style training scheme of these models where the entire video is always available to the model. Furthermore, Videollm-online [3] predicts narrations which are not fully informative of the action and therefore a mistake is detected very often leading to a very high mistake recall and a very low instruction completion detection accuracy.

Overall, when it comes to mistake detection, none of the zero-shot approaches performs well. Qwen2-VL-7B-Instruct [32] overestimates the occurrence of mistakes, leading to higher recall but low precision. Qwen2.5-VL-7B-Instruct [10] on the other hand is more precise in detecting mistakes, but with low overall F1 score. This weak performance can be attributed to the fact that detecting mistakes in the LIVECOOK benchmark is very challenging due to a variety of reasons. First is the fine-grained nature of many mistakes, *i.e.*, taking 1 teaspoon vs 1 tablespoon or 2 teaspoons of sugar, spilling flour on the kitchen counter etc, requires both fine-grained object recognition and action-recognition abilities. This is made additionally challenging by the fact that the model needs to look out of a diverse set of possible mistakes. Secondly, mistakes need to be detected as soon as they occur (similar to the instruction completions). This again exposes the limitations of current models in understanding scenes as they unfold in a streaming setup.

Finally, we report evaluation results of fine-tuned models in Tab. 2. Our LIVEMAMBA is trained on the LIVECOOK dataset and crucially on augmented counterfactual mistakes along with narration data from Ego4D [8] (details in the appendix), and uses a stronger vision encoder compared to Videollm-online. We see that training just on the LIVECOOK dataset is not enough as demonstrated by the weak performance of the LIVEMAMBA (w/o-Aug) baseline. This provides a strong clue to help understand the weak performance across models on the LIVECOOK benchmark: the lack of appropriate large-scale training data (with mistakes) for step-by-step task guidance.

4 REASONS FOR FAILURE AND CONCLUSION

We explore the challenge of enabling multi-modal LLMs to provide live, interactive step-by-step guidance. To enable evaluation (and training) on this challenging task, we introduce theLIVECOOK dataset and benchmark. It features densely annotated, timed instructions and feedback, including for mistakes, timestamped to visual occurrences. Extensive evaluation of state of the art multi-modal LLMs show that they struggle to provide live step-by-step task guidance. To understand the reason for this poor performance, we also experiment with the LIVEMAMBA model, a streaming multi-modal testbed designed for this task. In detail, LIVEMAMBA utilizes a"when-to-say" mechanism for live feedback and crucially a novel data augmentation for mistake recognition. The data augmentation scheme leads to significant boost in performance illustrating the lack of appropriate large-scale training data (with mistakes) a significant bottleneck for step-by-step task guidance.

REFERENCES

[1] Kirolos Ataallah, Xiaoqian Shen, Eslam Abdelrahman, Essam Sleiman, Deyao Zhu, Jian Ding, and Mohamed Elhoseiny. Minigpt4-video: Advancing multimodal llms for video understanding with interleaved visual-textual tokens. *arXiv preprint arXiv:2404.03413*, 2024.

[2] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020.

[3] Joya Chen, Zhaoyang Lv, Shiwei Wu, Kevin Qinghong Lin, Chenan Song, Difei Gao, Jia-Wei Liu, Ziteng Gao, Dongxing Mao, and Mike Zheng Shou. Videollm-online: Online video large language model for streaming video. In *CVPR*, 2024.

[4] Joya Chen, Ziyun Zeng, Yiqi Lin, Wei Li, Zejun Ma, and Mike Zheng Shou. Livecc: Learning video llm with streaming speech transcription at scale. *arXiv preprint arXiv:2504.16030*, 2025.

[5] Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, et al. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *CVPR*, 2024.

[6] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Antonino Furnari, Jian Ma, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, and Michael Wray. Rescaling egocentric vision: Collection, pipeline and challenges for epic-kitchens-100. In *IJCV*, 2022.

[7] An Yang et. al. Qwen2.5 technical report. *CoRR*, abs/2412.15115, 2024.

[8] Grauman et. al. Ego4d: Around the world in 3,000 hours of egocentric video. In *CVPR*, 2022.

[9] Marah I Abdin et. al. Phi-3 technical report: A highly capable language model locally on your phone. *CoRR*, abs/2404.14219, 2024.

[10] Shuai Bai et. al. Qwen2.5-vl technical report. *CoRR*, abs/2502.13923, 2025.

[11] Raghav Goyal, Samira Ebrahimi Kahou, Vincent Michalski, Joanna Materzynska, Susanne Westphal, Heuna Kim, Valentin Haenel, Ingo Fruend, Peter Yianilos, Moritz Mueller-Freitag, Florian Hoppe, Christian Thurau, Ingo Bax, and Roland Memisevic. The "something something" video database for learning and evaluating visual common sense. In *ICCV*, 2017.

[12] Kristen Grauman, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, Hao Jiang, Miao Liu, Xingyu Liu, Miguel Martin, Tushar Nagarajan, Ilija Radosavovic, Santhosh Kumar Ramakrishnan, Fiona Ryan, Jayant Sharma, Michael Wray, Mengmeng Xu, Eric Zhongcong Xu, Chen Zhao, Siddhant Bansal, et al. Ego4d: Around the world in 3, 000 hours of egocentric video. In *CVPR*, 2022.

[13] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *CoRR*, abs/2312.00752, 2023. doi: 10.48550/ARXIV.2312.00752. URL `https://doi.org/10.48550/arXiv.2312.00752`.

[14] Agrim Gupta, Piotr Dollár, and Ross B. Girshick. LVIS: A dataset for large vocabulary instance segmentation. In *CVPR*, 2019.

[15] Yang Jin, Zhicheng Sun, Kun Xu, Liwei Chen, Hao Jiang, Quzhe Huang, Chengru Song, Yuliang Liu, Di Zhang, Yang Song, et al. Video-lavit: Unified video-language pre-training with decoupled visual-motional tokenization. *arXiv preprint arXiv:2402.03161*, 2024.

[16] Junnan Li, Dongxu Li, Silvio Savarese, and Steven C. H. Hoi. BLIP-2: bootstrapping language-image pre-training with frozen image encoders and large language models. In *ICML*, 2023.

[17] KunChang Li, Yinan He, Yi Wang, Yizhuo Li, Wenhai Wang, Ping Luo, Yali Wang, Limin Wang, and Yu Qiao. Videochat: Chat-centric video understanding. *arXiv preprint arXiv:2305.06355*, 2023.

[18] Kunchang Li, Yali Wang, Yinan He, Yizhuo Li, Yi Wang, Yi Liu, Zun Wang, Jilan Xu, Guo Chen, Ping Luo, et al. Mvbench: A comprehensive multi-modal video understanding benchmark. In *CVPR*, 2024.

[19] Yanwei Li, Chengyao Wang, and Jiaya Jia. Llama-vid: An image is worth 2 tokens in large language models. In *European Conference on Computer Vision*, pp. 323–340. Springer, 2024.

[20] Bin Lin, Bin Zhu, Yang Ye, Munan Ning, Peng Jin, and Li Yuan. Video-llava: Learning united visual representation by alignment before projection. *arXiv preprint arXiv:2311.10122*, 2023.

[21] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *NeurIPS*, 2023.

[22] Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. Llava-next: Improved reasoning, ocr, and world knowledge, January 2024. URL `https://llava-vl.github.io/blog/2024-01-30-llava-next/`.

[23] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL `https://openreview.net/forum?id=Bkg6RiCqY7`.

[24] Muhammad Maaz, Hanoona Rasheed, Salman Khan, and Fahad Khan. Video-ChatGPT: Towards detailed video understanding via large vision and language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *ACL*, August 2024.

[25] Sunny Panchal, Apratim Bhattacharyya, Guillaume Berger, Antoine Mercier, Cornelius Böhm, Florian Dietrichkeit, Reza Pourreza, Xuanlin Li, Pulkit Madan, Mingu Lee, Mark Todorovich, Ingo Bax, and Roland Memisevic. What to say and when to say it: Live fitness coaching as a testbed for situated interaction. In *NeurIPS*, 2024.

[26] Rohith Peddi, Shivvrat Arya, Bharath Challa, Likhitha Pallapothula, Akshay Vyas, Bhavya Gouripeddi, Qifan Zhang, Jikai Wang, Vasundhara Komaragiri, Eric D. Ragan, Nicholas Ruozzi, Yu Xiang, and Vibhav Gogate. Captaincook4d: A dataset for understanding errors in procedural activities. In *NeurIPS*, 2024.

[27] Yale Song, Eugene Byrne, Tushar Nagarajan, Huiyu Wang, Miguel Martin, and Lorenzo Torresani. Ego4d goal-step: Toward hierarchical understanding of procedural activities. In *NeurIPS*, 2023.

[28] Yansong Tang, Dajun Ding, Yongming Rao, Yu Zheng, Danyang Zhang, Lili Zhao, Jiwen Lu, and Jie Zhou. Coin: A large-scale dataset for comprehensive instructional video analysis. In *CVPR*, 2019.

[29] Gemini Team. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *CoRR*, abs/2507.06261, 2025.

[30] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: A family of highly capable multimodal models. *CoRR*, abs/2312.11805, 2023.

[31] Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

[32] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024.

[33] 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 *CVPR*, 2025.

[34] Boqiang Zhang, Kehan Li, Zesen Cheng, Zhiqiang Hu, Yuqian Yuan, Guanzheng Chen, Sicong Leng, Yuming Jiang, Hang Zhang, Xin Li, Peng Jin, Wenqi Zhang, Fan Wang, Lidong Bing, and Deli Zhao. Videollama 3: Frontier multimodal foundation models for image and video understanding, 2025.

[35] Hang Zhang, Xin Li, and Lidong Bing. Video-llama: An instruction-tuned audio-visual language model for video understanding. In *EMNLP - System Demonstrations*, 2023.

[36] Bin Zhu, Bin Lin, Munan Ning, Yang Yan, Jiaxi Cui, HongFa Wang, Yatian Pang, Wenhao Jiang, Junwu Zhang, Zongwei Li, et al. Languagebind: Extending video-language pretraining to n-modality by language-based semantic alignment. *arXiv preprint arXiv:2310.01852*, 2023.

A Appendix

In the following we provide details of the annotation process of our LiveCook dataset and benchmark, details of the prompts used for zero-shot evaluation and training details of our LiveMamba model.

B LiveCook Annotation Details

Our LiveCook dataset and benchmark is built upon the CaptainCook4D dataset [26]. The CaptainCook4D dataset contains 384 videos in total. Each video records a person preparing a dish given a recipe, such as breakfast burritos and tomato mozzarella salad, from an egocentric view. Each video is associated with a graph-structured recipe[1], and annotated with temporal action segments: action descriptions with the corresponding starting time and ending time. If an action contains a mistake, then there is a description of the mistake. However, there is no timestamp annotation for the mistake. The average duration of action segments in CaptainCook4D is about 52.78 seconds , which is about 106 frames if using frame rate as 2. This also motivates us to use a light-weight Mamba based model to encode more frames given the hardware constraint.

The CaptainCook4D dataset includes 7 categories of mistakes: `preparation error`, `technique error`, `measurement error`, `temperature error`, `timing error`, `order error`, and `missing steps`. We annotate the timestamps of the mistakes when they just happened. We remove some noisy mistake annotations in CaptainCook4D, and annotate for all mistake categories except for `order error` and `missing steps`. Tab. 3 shows the numbers of mistake instances per mistake category. Note, we show the mistake statistics here for a complete description of the LiveCook dataset and benchmark. This mistake category information is not used in our experiments presented in the paper.

Our LiveCook features with a step-by-step plan, and timestamped instructions and feedback for each video recording. We assume that the user always tries to follow the given instruction. That is, there is no action order error. Given a video, we first sort the actions by their starting time in ascending order (primary key) and ending time in descending order (secondary key). Based on this order, we build the step-by-step plan. Usually one action description forms one step in the plan. In cases where one action is temporally contained in another action (actions performed in parallel), we group those action descriptions into one step. Accordingly, we create action groups, where one step in the plan corresponds to one action group. The first step in the plan provides the contents for the next instruction to be given.

Given a video, we treat the first action's starting time as the video starting time and the first instruction is given at that time. Once an action finishes, its description is removed from the step-by-step plan. Once all the actions in the current action group finish, the current step in the plan will be empty. We remove the empty step in the plan and give the instruction for the next step. The instruction is a sentence containing all the information in the action descriptions in the step. Once an action finishes successfully, we acknowledge the success via summarizing the description of the just finished action, and using the words like successfully, correctly, properly, and etc. This type of feedback is given at the action's ending time. If an action contains a mistake, we give the feedback regarding the mistake at our annotated timestamp. The feedback is a sentence pointing out the mistake based on the mistake description given in CaptainCook4D. Once all the steps in the initial plan are completed, we output the feedback "`You have finished all the steps.`" to acknowledge the completion. Fig. 3 shows data samples. Whenever an action finishes, it is removed from the step-by-step plan. The examples show that recognizing mistakes requires fine-grained understanding of user actions and objects in the scene across long time-horizons.

C Zero-shot Evaluation

Here we provide the prompts used for zero-shot evaluation. We begin with a more detailed description of the prompting strategy in Section 2 and 3.1 of the main paper.

[1]Examples can be found at `https://captaincook4d.github.io/captain-cook/recipe.html`.

Table 3: Mistake category statistics.

Split	Preparation Err.	Technique Err.	Measurement Err.	Temperature Err.	Timing Err.
Training	198	224	159	27	78
Validation	63	79	68	10	37
Testing	110	156	113	17	49

Promoting Strategy. This strategy involves prompting the model at regular intervals to detect both successful completions of instructions and mistakes. However prompting after every input frame is not computationally feasible. To balance accuracy and compute requirements, we prompt the models at an interval of 5 seconds. First, the model is asked if the user has completed the current instruction (obtained from the recipes in the LIVECOOK benchmark). We use the following prompt for the Gemini-2.5-Flash [31], Qwen2.5-VL-7B-Instruct [10], Qwen2-VL-7B-Instruct [32], VideoLLaMA3-7B [34] models:

Gemini-2.5-Flash/ Qwen2x-VL-7B-Instruct / VideoLLaMA3-7B: Check if instruction complete

You are an expert cooking assistant helping a person cook. The person is provided with an instruction and your task is to check if the instruction has been completed.

##INSTRUCTIONS:
The person has been instructed to: [instruction].
If the person has completed the instruction answer "yes" else answer "no". DO NOT OUTPUT ANY OTHER TEXT.

In case of VideoChat2 [18] model, we use the following prompt:

VideoChat2: Check if instruction complete

You are an expert cooking assistant helping a person cook. The person is provided with an instruction and your task is to check if the instruction has been completed. You must check the video content very closely and confirm if the instruction has been completely followed and finished.

##INSTRUCTIONS:
The person has been instructed to: [instruction].
If the person has completed the instruction answer "yes" else answer "no". Answer "yes" ONLY IF the instructed action is completed in the video, otherwise answer "no".

In case of the Video-LLaVA [36, 20], Video-ChatGPT [24], LLaVA-NeXT [22] as they do not accept system messages we use the following simplified prompt:

Video-LLaVA / Video-ChatGPT / LLaVA-NeXT : Check if instruction complete

The person has been instructed to: [instruction].
If the person has completed the instruction answer "yes" else answer "no".

If the model answers "yes", then we move on to the next instruction in the recipe. If not, then we ask the model to check for mistakes.

For VideoLLM-Online – since the model is trained to narrate videos, it can not detect completion and mistakes out of the box. Therefore, we use a helper LLM (Phi-3-mini-4k-Instruct) to help detect completion and mistakes given narrations. We use the following prompts,

VideoLLM-Online: Check if instruction is complete

VideoLLM-Online: Please narrate the video in real-time.
Helper-LLM (Phi-3-mini-4k-Instruct): You are an intelligent chatbot that is judging another system which narrates human cooking videos. Given a high level action instruction and a list of narrations generated from the system, your job is to decide if the narration is correct and shows completion of the instruction. Answer 'yes' if the instruction is completed otherwise output 'no'.
Instruction: [instruction]
Narrations: [videollm-online narrations]

In the offline (turn-based) evaluation scheme used in CaptainCook4D [26], multi-modal LLMs are prompted with sequence of questions per error category to detect mistakes. However, such a scheme is infeasible in case of streaming setups due to high computational costs. Therefore, to enable mistake detection in a single inference step, we design a prompt that concisely explains the possible mistakes that are possible while following a given instruction. The model can then use this information to recognize possible mistakes. In detail, for a given instruction we find similar instructions in the training set and use Qwen-2.5-32B-Instruct [7] to summarize the possible mistakes. The following are some examples:

Instruction: Now place 8-inch flour tortilla on cutting board.

When guiding the user through this step, the cooking instructor should watch out for these potential mistakes:

1. Users might toast or heat the tortilla before placing it on the cutting board.
2. Users may place the tortilla on a plate instead of a cutting board.
3. Users could use an unclean surface instead of a clean cutting board.

Instruction: Now add 1/4 tsp salt to a bowl.

When guiding the user through this step, the cooking instructor should watch out for these potential mistakes:

1. Spilling salt while measuring or adding it.
2. Adding too much salt, specifically 1/2 tsp instead of the required 1/4 tsp.
3. Confusing 1/3 tablespoon with 1/3 teaspoon.
4. Accidentally adding the salt to the pan rather than the bowl.

We then use these "mistake summaries" to prompt the multi-modal LLM. For Gemini-2.5-Flash [31], Qwen2.5-VL-7B-Instruct [10], Qwen2-VL-7B-Instruct [32], VideoLLaMA3-7B [34], VideoChat2 [18] we use the following prompts:

Gemini-2.5-Flash / Qwen2x-VL-7B-Instruct / VideoLLaMA3-7B/ VideoChat2: Check if instruction complete

You are an expert cooking assistant who is observing a person who is provided with step by step instructions for cooking. You should look out for mistakes made by the person.

##INSTRUCTIONS:
The person is trying to complete the following instruction: [instruction].
This is how you can check for mistakes: [mistake summary].
Your task is to check if the person has already made a mistake.
Note that the person may not have completed the provided instruction, that is, the person may have only partially completed the provided instruction.
The answer should be "yes" or "no". In case of yes, please provide a concise feedback to the person describing the mistake (i.e. Yes. `<feedback>`.). Directly address the person.

In case of Video-LLaVA [36, 20], Video-ChatGPT [24], LLaVA-NeXT [22] models, they do not support complex prompts as they were mainly designed for question answering tasks. They we employ an alternative strategy where we ask the model to narrate the video in 30 second chunks. Then we use the (concatenated) past narrations since the last instruction completion to prompt the model to look for mistakes.

Video-LLaVA / Video-ChatGPT / LLaVA-NeXT : Check if instruction complete

The person has been instructed to: [instruction].
ill now the person has done the following: [past_narrations].
Your task is to check if the person has made a mistake.
The answer should be "yes" or "no". In case of "yes", please provide a feedback to the user describing the mistake. Directly address the person.

For VideoLLM-Online, we use a similar strategy as done in completion detection. We feed VideoLLM-Online's narration to a helper LLM to get mistake detection. We use the following prompts,

VideoLLM-Online: Check if instruction is complete

VideoLLM-Online: Please narrate the video in real-time.
Helper-LLM (Phi-3-mini-4k-Instruct): You are an intelligent chatbot that is judging another system which narrates human cooking videos. Given a high level action instruction and a list of narrations generated from the system, your job is to decide if a mistake has been made.
The user has be instructed to do the following: [instruction]
Till now the person has done the following: [videollm-online narrations]
Your task is to check if the person has made a mistake.
The answer should be yes or no. In case of yes, please provide a feedback to the user describing the mistake. Directly address the person.

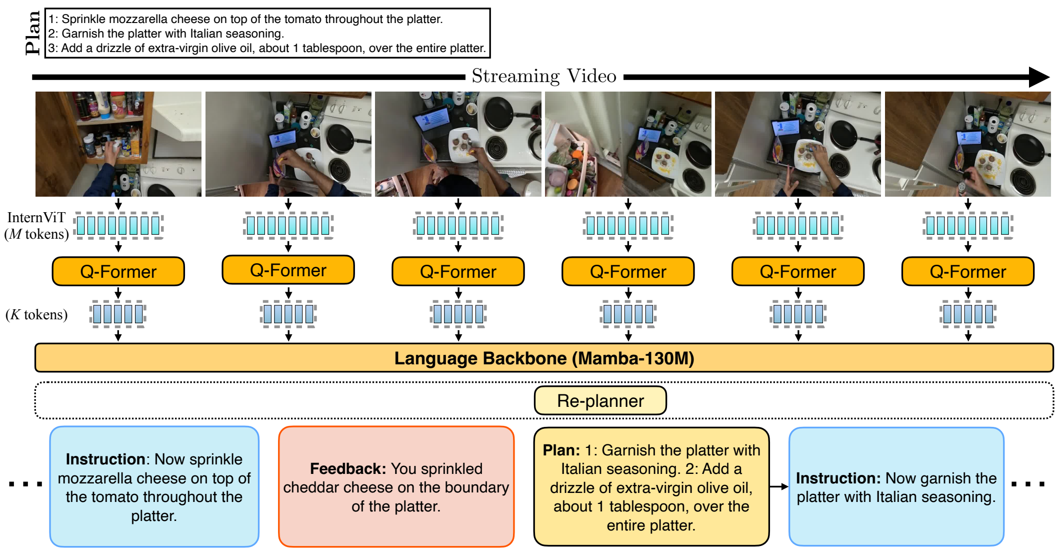

Plan
1: Sprinkle mozzarella cheese on top of the tomato throughout the platter.
2: Garnish the platter with Italian seasoning.
3: Add a drizzle of extra-virgin olive oil, about 1 tablespoon, over the entire platter.
Streaming Video
InternViT (M tokens)
Q-Former
(K tokens)
Language Backbone (Mamba-130M)
Re-planner
Instruction: Now sprinkle mozzarella cheese on top of the tomato throughout the platter.
Feedback: You sprinkled cheddar cheese on the boundary of the platter.
Plan: 1: Garnish the platter with Italian seasoning. 2: Add a drizzle of extra-virgin olive oil, about 1 tablespoon, over the entire platter.
Instruction: Now garnish the platter with Italian seasoning.

Figure 2: Our LIVEMAMBA model architecture. The input video stream is processed by an InternViT vision head which produces M tokens, and is then reduced to K tokens by a Q-Former. The language backbone produces feedback and invokes the Re-planner if necessary before the next instruction.

D LIVEMAMBA TRAINING AND DATA AUGMENTATION

We illustrate our LIVEMAMBA model architecture in Fig. 2. We now provide details of our training (and data augmentation) scheme, which is composed of two stages.

D.1 Pre-training

This stage aligns the Q-Former's vision embeddings with the language backbone's text embeddings by training only the Q-Former adapter. We use a diverse set of image and video datasets to instill two key visual skills: First, for object grounding, LiveMamba is trained on image datasets like LVIS [14] (for diverse household objects) and on video data using VISOR annotations from EPIC-KITCHENS [6]. Tasks include image/video captioning, object recognition, and bounding box prediction. Second, for fine-grained action understanding, LiveMamba learns action recognition from SSv2 [11] and video narration from EPIC-KITCHENS and Ego4D. Narration helps ground the model to actions in these large-scale egocentric datasets, particularly cooking activities.

In detail, the data follows a question answer format, where the questions are of the following format (following [21]):

LVIS / EPIC-KITCHENS: Grounding Questions

- Please provide the bounding box coordinates for the ______., A: ______
- Where is the ______ located in the image?,A: It is located at ______.
- What are the coordinates of the bounding box encompassing the ______?, A: ______.
- Can you pinpoint the ______ in the image by giving me its bounding box coordinates?, A: Yes, it is at ______.
- Where is the ______ in the image? Give me the bounding box coordinates., A: ______.
- I need the top-left and bottom-right corners of the ¡1¿'s bounding box. What are they?, A: ______.
- Could you identify the ______ in the image and tell me its location using bounding box coordinates?, A: Yes, it is at ______.

- Describe the image concisely., A: ______.
- Provide a brief description of the given image., A: ______.
- Offer a succinct explanation of the picture presented., A: ______.
- Summarize the visual content of the image., A: ______.
- Give a short and clear explanation of the subsequent image., A: ______.
- Share a concise interpretation of the image provided., A; ______.
- Present a compact description of the photo's key features., A: ______.
- Relay a brief, clear account of the picture shown., A; ______.
- Render a clear and concise summary of the photo., A: ______.
- Write a terse but informative summary of the picture., A: ______.
- Create a compact narrative representing the image presented., A: ______.

For action recognition tasks on SSv2 [11], we use the following format:

SSv2: Action Recognition Questions

- Describe the action I performed in this video in detail and name the objects that the I interacts with?, A: ______.
- Can you provide a step-by-step description of the action in the video which includes the specific objects that I touched or manipulated?, A: ______.
- Tell me a description of the action performed by me in the video that includes the names of any items that I interact with., A: ______.
- What action happens in the video and what objects are involved in the action?, A: ______.
- Describe my actions and the objects that I come across., A: ______.

For narration tasks on EPIC-KITCHENS we convert the action descriptions to second person format, *e.g.*, "pick up plate" to "You picked up a plate." For Ego4D, we employ the narrations used by [3].

D.2 Fine-tuning

During the fine-tuning phase the LiveMamba model is trained to provide the appropriate instructions and feedback at the appropriate time, using the ¡vision¿ and ¡feedback¿ special tokens as described above. At this stage, both the Q-Former's based adapter and the language backbone is trained. The LiveMamba needs to recognize the successful completion of instructions and mistakes. To this end, in addition to training on the step-by-step instructions and feedbacks from our LiveCook dataset, we apply several augmentations as described below.

Temporal Augmentation. To maintain temporal accuracy of feedbacks while dealing with long videos in the LiveCook benchmark, we introduce temporal jittering during training. Specifically, we jitter the starting timestamp of each instruction by a constant $\pm$K seconds. This temporal jittering deals that fact that predictions by autoregressive models, *e.g.*, our LiveMamba, can accumulate errors. Jittering the starting timestamp of an instruction ensures that the LiveMamba can successfully predict feedbacks irrespective of the previous accumulated error. In practice, we find $K = 30$ to work well.

Instruction Completion Augmentation. To help the LiveMamba recognize the successful completion of instructions, we augment our training set by converting videos from the EPIC-KITCHENS and Ego4D datasets to the step-by-step instruction and feedback format of our LiveCook dataset. In detail, we consider the action descriptions in the EPIC-KITCHENS dataset and the Ego4D Goal-Step datasets and use a Qwen2.5-8B model [7] to convert these action descriptions to instructions. We provide the instruction at the action start time and feedback message at the action end timestamp. As these datasets do not contain any mistakes, the feedback messages acknowledge the successful completion of the instruction.

Counterfactual Mistake Augmentation. Recognizing mistakes and providing timely feedback is a key challenge of our LiveCook benchmark. To this end, we formulate a novel data augmentation scheme to generate (counterfactual) mistakes in the EPIC-KITCHENS and Ego4D datasets. First, we convert the action description to plausible grounded counterfactual action descriptions. These grounded counterfactual action descriptions are used to generate instructions and thus construct scenarios where the user tries to follow the given instruction but makes a mistake.

In detail, we utilize the Ego4D goalstep annotations [27] and the FHO annotations [8]. The FHO annotations largely include fine-grained short duration actions while the goalstep annotations include longer-ranged actions more closely aligned with the recipe steps in our LiveCook benchmark. To create temporally localized counterfactual mistakes, we first find the FHO actions that are included within each goalstep action. Then we ask Qwen-2.5-32B-Instruct [7] identify the "critical" FHO action for each goalstep action, *e.g.*, for the goalstep action `wash green beans in water`, the critical FHO identified by Qwen-2.5-32B-Instruct is `washes green bean` (from the following FHO actions within the goalstep action: `collects green bean`, `puts green beans on the chopping board`, `puts green beans in cooking pan`, `...`, `picks green beans`, `opens tap`, `washes green bean`, `...`). We then ask the Qwen-2.5-32B-Instruct to construct two counterfactual actions. Firstly, by changing the noun to an alternative noun in the scene, *i.e.*, `washes green bean` to `washes carrots`. The list of nouns in the scene is generated by identifying objects in the scene using DETR [2]. Secondly, by proposing an alternative verb, *i.e.*, `washes green bean` to `mash green bean`. Then we use this counterfactual action to create an instruction and feedback pair. We use the point-of-no-return timestamp [8] for the mistake feedback.

In case of the EPIC-KITCHENS dataset, most actions are of short duration (¡10 seconds long). We directly use Qwen-2.5-32B-Instruct to construct two counterfactual actions by using an alternative noun in the scene and by an alternative verb as described above. As the actions are short we use the end of action timestamp to generate the (mistake) feedback.

D.3 Training and Inference Hyperparameters.

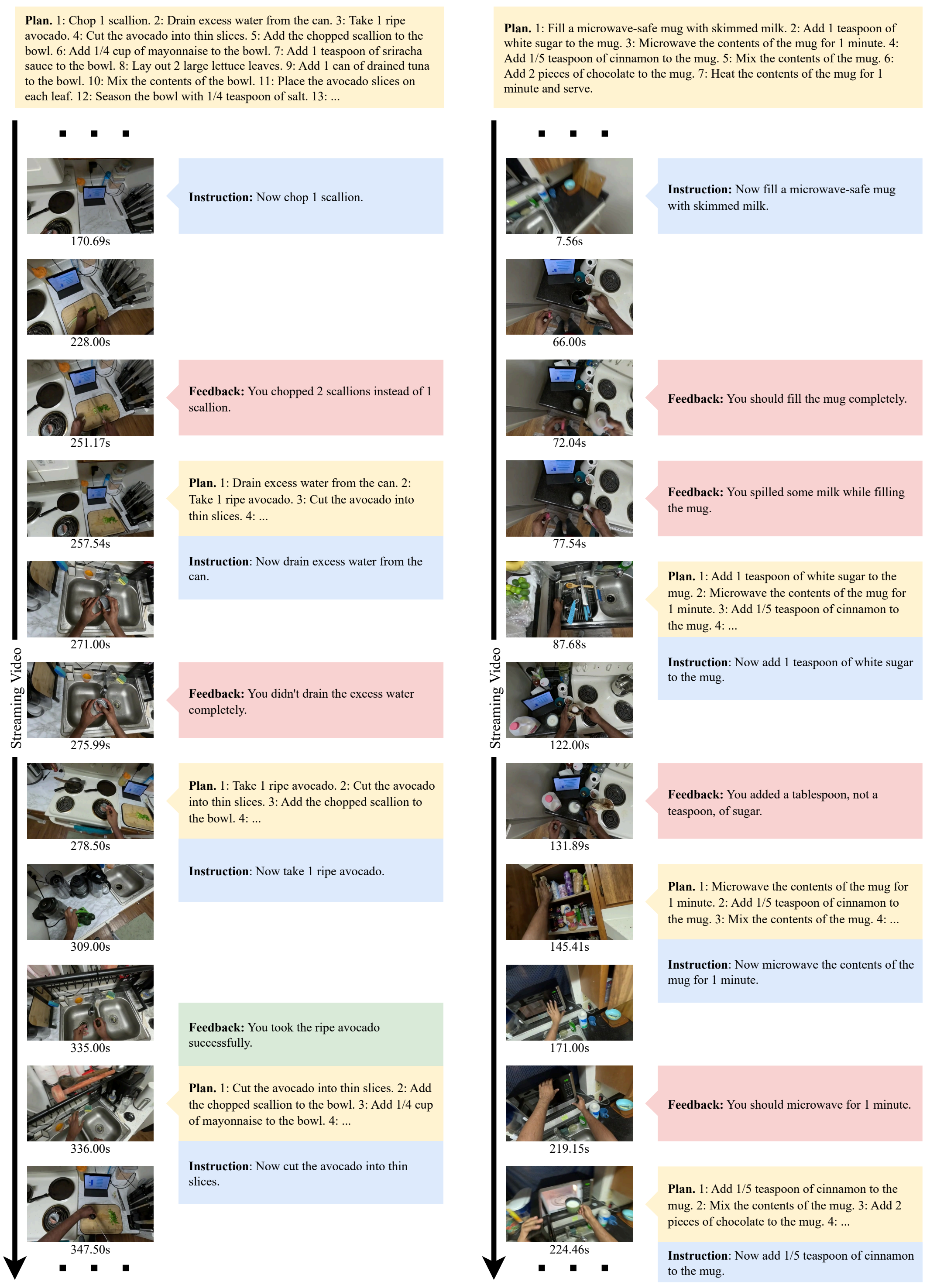
We use input video resolution of 448×448 at 2 fps. The InternViT-300M-448px-V2_5 vision head produces $N = 1025$ tokens (including the CLS token) per input frame. We use the mechanism outlined in VisionZip [33] to reduce the number of tokens to 256. Then, our Q-Former reduces this further to $K = 32$ tokens.

In addition to the vision features, the Mamba-130M language backbone of the LiveMamba model is prompted with the following prompt:

LiveMamba: Interactive Inference (Language Backbone)

You are an expert cooking assistant that is helping a person cook the following step by step recipe: [recipe_steps].
You just provided the following instruction to the person: [last_instruction].
Now watch the video and provide the appropriate success or failure messages.

The LiveMamba model is trained using 8 Nvidia H100 GPUs. We use the AdamW [23] optimizer. During the pre-training phase, we train only the Q-Former and the LiveMamba model is trained using a learning rate of 1×10^{-5} for 200k iterations. We again use a learning rate of 1×10^{-5}, for 120k iterations. During the fine-tuning phase, we train on single recipe steps and clip the maximum length to 3 minutes. During inference, we re-initialize the LiveMamba model after every recipe step.



Plan. 1: Chop 1 scallion. 2: Drain excess water from the can. 3: Take 1 ripe avocado. 4: Cut the avocado into thin slices. 5: Add the chopped scallion to the bowl. 6: Add 1/4 cup of mayonnaise to the bowl. 7: Add 1 teaspoon of sriracha sauce to the bowl. 8: Lay out 2 large lettuce leaves. 9: Add 1 can of drained tuna to the bowl. 10: Mix the contents of the bowl. 11: Place the avocado slices on each leaf. 12: Season the bowl with 1/4 teaspoon of salt. 13: ...

Instruction: Now chop 1 scallion.

Feedback: You chopped 2 scallions instead of 1 scallion.

Plan. 1: Drain excess water from the can. 2: Take 1 ripe avocado. 3: Cut the avocado into thin slices. 4: ...

Instruction: Now drain excess water from the can.

Feedback: You didn't drain the excess water completely.

Plan. 1: Take 1 ripe avocado. 2: Cut the avocado into thin slices. 3: Add the chopped scallion to the bowl. 4: ...

Instruction: Now take 1 ripe avocado.

Feedback: You took the ripe avocado successfully.

Plan. 1: Cut the avocado into thin slices. 2: Add the chopped scallion to the bowl. 3: Add 1/4 cup of mayonnaise to the bowl. 4: ...

Instruction: Now cut the avocado into thin slices.

Plan. 1: Fill a microwave-safe mug with skimmed milk. 2: Add 1 teaspoon of white sugar to the mug. 3: Microwave the contents of the mug for 1 minute. 4: Add 1/5 teaspoon of cinnamon to the mug. 5: Mix the contents of the mug. 6: Add 2 pieces of chocolate to the mug. 7: Heat the contents of the mug for 1 minute and serve.

Instruction: Now fill a microwave-safe mug with skimmed milk.

Feedback: You should fill the mug completely.

Feedback: You spilled some milk while filling the mug.

Plan. 1: Add 1 teaspoon of white sugar to the mug. 2: Microwave the contents of the mug for 1 minute. 3: Add 1/5 teaspoon of cinnamon to the mug. 4: ...

Instruction: Now add 1 teaspoon of white sugar to the mug.

Feedback: You added a tablespoon, not a teaspoon, of sugar.

Plan. 1: Microwave the contents of the mug for 1 minute. 2: Add 1/5 teaspoon of cinnamon to the mug. 3: Mix the contents of the mug. 4: ...

Instruction: Now microwave the contents of the mug for 1 minute.

Feedback: You should microwave for 1 minute.

Plan. 1: Add 1/5 teaspoon of cinnamon to the mug. 2: Mix the contents of the mug. 3: Add 2 pieces of chocolate to the mug. 4: ...

Instruction: Now add 1/5 teaspoon of cinnamon to the mug.

Figure 3: Data samples from LIVECOOK. Left: the user prepares spicy tuna avocado wraps. Right: the user prepares spiced hot chocolate.