Pretrained Language Models as Visual Planners for Human Assistance

Dhruvesh Patel[1,2] Hamid Eghbalzadeh[1] Nitin Kamra[1]
Michael Louis Iuzzolino[1] Unnat Jain[1*] Ruta Desai[1*†]
[1]Meta [2]UMass Amherst

https://github.com/facebookresearch/vlamp

Abstract

To make progress towards multi-modal AI assistants which can guide users to achieve complex multi-step goals, we propose the task of 'Visual Planning for Assistance (VPA)'. Given a goal briefly described in natural language, e.g., "make a shelf", and a video of the user's progress so far, the aim of VPA is to obtain a plan, i.e. *a sequence of actions such as "sand shelf", "paint shelf", etc. to achieve the goal. To address the challenges that VPA brings,* i.e. *handling long video history, and arbitrarily complex action dependencies, we decompose VPA into video action segmentation and forecasting. We formulate the forecasting step as a multi-modal sequence modeling problem and present **V**isual **La**nguage **M**odel based **P**lanner (VLaMP), which leverages pre-trained LMs as the sequence model. We demonstrate that VLaMP performs significantly better than baselines w.r.t all metrics that evaluate the generated plan.*

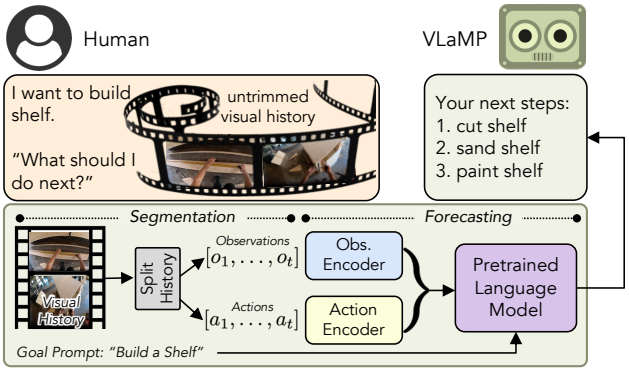


Figure 1: Given a user's goal described in natural language and corresponding visual history[1] depicting the user's progress till time t, the aim of Visual Planning for Assistance is to plan a sequence of actions aimed at achieving the goal. Our approach entails multi-modal sequence modeling using pre-trained video segmentation and language models.

1. Introduction

Imagine assembling a new piece of furniture or making a new recipe for a dinner party. To achieve such a goal, you might follow a manual or a video tutorial, going back and forth as you perform the steps. Instead, imagine an assistive agent capable of being invoked through natural language, having the ability to understand human actions, and providing actionable multi-step guidance for achieving your desired goal. To develop such multi-modal agents, we propose a new and intuitive learning task – *Visual Planning for Assistance (VPA)*.

As illustrated in Fig. 1, given a user-specified goal in natural language and corresponding video observations depicting the user's progress towards this goal, the VPA objective is to generate the ordered sequence of next actions towards achieving the goal. Focusing on the ubiquitous category of multi-step stream of actions involved in human activities, we based VPA off instructional videos of procedural activities like cooking, repair, assembly, etc., from YouTube.

Given a procedural activity, the assistant must estimate the user's progress from a long video. Then, conditioned on the user's progress, the assistant must generate and recommend a *valid plan* of actions. To overcome these challenges, we formulate an approach for VPA that is based on video action segmentation and a transformer-based neural sequence modeling, where the former allows us to deal with long video history, while the latter can handle arbitrary sequential constraints [18, 8]. Furthermore, our formulation enables leveraging state-of-the-art pre-trained transformer based language models (PTLMs), which may contain useful priors about action-action similarity, action-goal association, and action ordering [2, 7, 10]. Our model, which we call Visual Language Model Planner (VLaMP), conditions the generated plan onto the visual history by using a transformer-based mapper network that projects embeddings corresponding to visual history into the input space of

*equal mentoring †corresponding author

[1]The images in the Fig. 1 and Fig. 2 are from Ego4D[5], and are for the purpose of illustration only.

the LM. We show that VLaMP performs significantly better than the baselines. [2]

2. Visual Planning for Assistance

In this section, we introduce Visual Planning for Assistance (VPA), include the definition of VPA, and describe the evaluation protocol. Firstly, the following two intuitive inputs are given to any model performing VPA.

Goal Prompt (G). The natural language description (in short phrase) of the user's goal. Ex: "build a shelf", "change a tire", etc.

Visual History (V_t). An untrimmed video that provides context about the user's progress towards a goal from the start till time, say t. Assume that V_t contains k actions pertaining to the goal, then VPA doesn't have access to these steps or k and must work with V_t directly.

The objective of VPA is to generate a *plan* $\mathcal{T} = (a_{k+1}, \ldots, a_{k+l})$, a sequence of actions that should be executed (in the next steps) to assist the user in achieving the goal G, where a_i are represented in natural language but come from a closed set $\mathcal{A}$.

Evaluation using Open-Sourced Video Data. We leverage existing datasets CrossTask [20] and COIN [13] for VPA, based on the following three requirements: (1) Rich diversity of activities from multiple domains, (2) goal-oriented activities consisting of long sequences of actions, and (3) Action annotations from a fixed closed set of actions.

Metrics. The planning performance of a VPA model is measured by comparing the generated plan $\hat{\mathcal{T}} = (\hat{a}_{k+1}, \ldots, \hat{a}_{k+l})$ to the ground truth plan $\mathcal{T}$ for l actions in the future, given V_t and G. Here $\hat{a}_{k+i}$ denotes the prediction for the $k+i$-th step given history till k-th step. We use the following metrics, listed in decreasing order of strictness: *success rate* (SR), *mean accuracy* (mAcc), *mean intersection over union* (mIOU), and accuracy of predicting the next action *i.e.* nAcc. Complete details of the metrics are included in Appendix J.

3. Visual LM Planner

In this section we describe our approach for VPA, called **V**isual **La**nguage **M**odel based **P**lanner (VLaMP), consisting of a segmentation module and PTLM based sequence prediction module.

3.1. Planning with Segmentation and Forecasting

We define π as a goal-conditioned, multi-modal sequence prediction problem $\pi = P(a_{k+1}, a_{k+2}, \cdots \mid V_t, G)$. However, due to the high-dimensional state space of untrimmed video V_t, and availability of limited data to learn the distribution over valid action sequences, working with π directly is challenging. So, as shown in Eq. 1, we *decompose* π into two modules: (1) **video segmentation** module, which converts the untrimmed video history V_t into a sequence of video segments *i.e.* a segment history $S_k = (s_1, \ldots, s_k)$, where each segment corresponds to an action a_i that occurred in the video; (2) **forecasting** module, which transforms the output of the segmentation module and generates the plan.

$$\pi = \sum_{S_k} \underbrace{P(a_{k+1}, a_{k+2}, \cdots \mid S_k, G)}_{\text{Forecasting}} \underbrace{P(S_k \mid V_t)}_{\text{Segmentation}}, \quad (1)$$

But, since the summation in Eq. 1 is intractable, we use it only as the guiding expression to formulate the input and output of both modules, which are described in the following subsections.

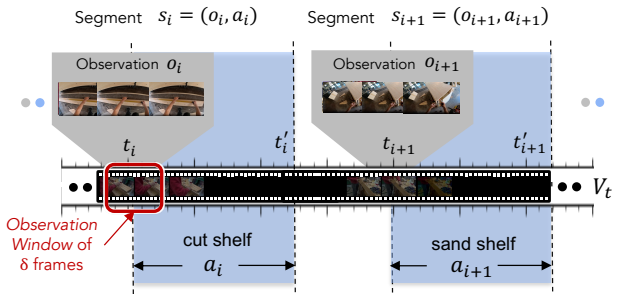


Figure 2: **VLaMP – Segmentation Module.** The untrimmed visual history V_t is converted into segments, each consisting of observation o_i and the action a_i. The observation o_i is the collection of video frames of δ seconds around the start time stamp t_i of the corresponding action a_i. Two such segments are shown here.

3.2. Segmentation Module

This module splits the untrimmed video history V_t into segment history $S_k = ((o_1, a_1), \ldots, (o_k, a_k))$ of multiple segments, each segment corresponding to an action. Here $A_k = (a_1, \ldots, a_k)$ and $O_k = (o_1, \ldots, o_k)$ are the action and observation history, respectively as illustrated in Fig. 2 (see Appendix K for details).

3.3. Forecasting Module

The usefulness of π's decomposition expressed in Eq. (1) becomes apparent now. Modeling the segmentation module's output as segment history S_k, where each segment consisting of action and observations, allows writing the output of the forecasting module in an autoregressive man-

[2] To enable use of our formulation as a benchmark for Visual Planning for Assistance, we release the data processing code, as well as metrics and model implementation.

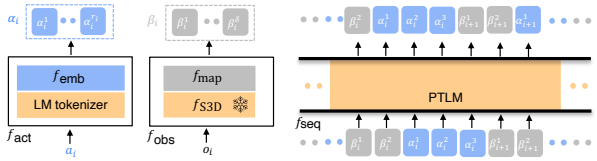


Figure 3: **VLaMP – Forecasting Module.** As shown in left and middle, actions and observations obtained from the segmentation module are encoded using appropriate modality encoders. The observation encoder leverages pretrained video encoder for observations, while also learning a mapper that aligns the representations from observations with actions. As shown on the right, VLaMP uses a joint sequence model on top of interleaved action (blue) and observation (gray) representations to forecast autoregressively the next representation.

ner:

$$P(a_{k+1}, a_{k+2}, \cdots \mid o_1, a_1, \ldots, o_k, a_k, G) = \prod_{i>0} \sum_{o_{k+i}} P(o_{k+i}, a_{k+i} \mid o_1, a_1, \ldots, o_k, a_k, G). \quad (2)$$

We instantiate the forecasting module for π using a pretrained transformer-based LM. Next we present the details of the encoders for the two modalities and the LM based sequence model.

Action encoder (f_{act}) Each action a_i in A_k is encoded by f_{act} and the output is denoted by α_i. Concretely, token embeddings are expressed as:

$$(\alpha_1, \ldots, \alpha_k) = (f_{\text{act}}(a_1), \ldots, f_{\text{act}}(a_k)), \text{where}$$
$$\alpha_i = (\alpha_i^1, \ldots, \alpha_i^{r_i}) \in \mathbb{R}^{r_i \times d} \quad (3)$$

As we illustrate in Fig. 3 (left), each action a_i is tokenized into r_i tokens using appropriate tokenizer for the LM, the tokens are indexed using the vocabulary of the LM, and are represented using an embedding lookup from the token embeddings of the LM to produce α_i. Here, r_i is the number of tokens and d is the dimensionality of token embeddings.

Observation encoder (f_{obs}). Recall, the visual observation history O_k comprises of o_i corresponding to action a_i, each of δ frames. As illustrated in Fig. 3 (middle), we transform each o_i employing the widely-adopted S3D backbone [16] f_{S3D^*} ($*$ denotes backbone is frozen). We must project visual encodings to a shared latent space of action (language) embeddings described before (α_i). To this end, we map S3D features via a trainable transformer mapper f_{map}. Concretely,

$$(\beta_1, \ldots, \beta_k) = (f_{\text{obs}}(o_1), \ldots, f_{\text{obs}}(o_k)), \text{where}$$
$$\beta_i = (\beta_i^1, \ldots, \beta_i^\delta) \in \mathbb{R}^{\delta \times d} \text{ and } f_{\text{obs}} = f_{\text{S3D}^*} \circ f_{\text{map}} \quad (4)$$

Overall, as we show in Fig. 3 (right), the resultant encoded sequence of representation for S_k is thus

$$f_{\text{enc}}(S_k) = (\beta_1, \alpha_1, \ldots, \beta_k, \alpha_k) = H_k,$$

Sequence model (f_{seq}). Given the above encoding for the segment history S_k, the role of the sequence model is to predict a representation of the next token, that would in return enable VLaMP plan generation capabilities for VPA. Importantly, in the process of generating sequence of future actions autoregressively, we would also need to generate the representations of 'future observations'. Therefore, as we shown in Fig. 3, our sequence model, which consists of the transformer layers of a PTLM, also produces representations for vision (in addition to the necessary action tokens).

3.4. Training

The joint training of the segmentation and forecasting modules following Eq. (1) is intractable. But, by exploiting the availability of unpaired training data, we approximate Eq. (1) by feeding in the output of the segmentation module to the forecasting module and training them separately, each on their respective labeled data. The video-action segmentation model is trained utilizing the VideoCLIP setup [17], where in the segmentation model performs classification to predict the action for each second of the video. The forecasting model is trained by adopting the next representation prediction objective. Unlike vanilla LM pretraining, however, we also need to train for predicting visual representations in addition to text (action). Therefore, we use two different losses L_{act} and L_{obs} for text and visual representations respectively. Specifically, L_{act} is the conventional cross-entropy loss over the LM's vocabulary V_{LM} for the action representations while L_{obs} is the mean-squared error between the predicted and the ground truth observation representations. The total loss is the sum of both the loss terms.In order to have a stable training, we use ground truth action history to construct S_k (and subsequently H_k) instead of the output of the segmentation module. Appendix provides further details on loss and optimizers for training.

Inference. The inference is performed using a beam-search like algorithm that works at the representation level and allows multi-token visual and text representations. The detailed algorithm is presented in Appendix C.

4. Experiments

We instantiate VLaMP's segmentation module utilizing VideoCLIP [17] fine-tuned on COIN and CrossTask; and the sequence model f_{seq} (in the forecasting module) by GPT2 [11, 15].

Data. For a video V with goal G, both CrossTask [20] and COIN [13] provide annotations of the form

$\{a_k, (t_k, t'_k)\}_{k=1}^{K}$, where a_k are the actions in the video, and t_k (resp. t'_k) are the start (resp. end) timestamps a_k. Given an annotated video consisting of K steps, we generate $K - l$ examples, each with input $x_k = (G, V_{t_k})$ and output $y_k = (a_{k+1}, \ldots, a_{k+l})$, for $k = 1, \ldots K - l$ (leaving at least l steps to predict in each example). [3] Therefore from M videos, we generated $N = \sum_{m=1}^{M}(K_m - l)$ examples, where K_m is the number of steps in the m-th video, forming a dataset $\mathcal{D} = \{x^{(j)}, y^{(j)}\}_{j=1}^{N}$ suitable for VPA (total number of samples in shown in Table **??**).

Baselines. As a first step towards benchmarking, we utilize a random baseline, and additionally adopt a variety of strong goal-conditioned models. The procedure planning task is most relevant task from the literature to VPA, therefore, we adapt [4] the widely used DDN model introduced by Chang *et al*. [4], since it is an established model in many benchmarks in prior works [1, 19, 12], hence, is chosen for our experiments. The key details regarding our baselines are as follows [5]:

• *Random*: Predicts the plan by picking all l actions uniformly randomly from the set of all actions $\mathcal{A}$.

• *Random w/ goal*: A stronger baseline; for each goal G, we allow privilege access to a set of applicable actions to that goal $\mathcal{A}_G \subseteq \mathcal{A}$, and predicts the plan by randomly picking actions from the restricted set.

• *Dual Dynamics Network (DDN)* [4]: We keep DDN's network structure but use Algo. 1 for inference on VPA.

4.1. Results

In the following, we include quantitative findings of benchmarking methods on two video data sources (Tab. 1). More quantitative results, ablations, error analysis, and qualitative results are included in the Appendix I.

Improved performance across video datasets. As we show in Tab. 1, VLaMP significantly outperforms the baselines. DDN that is customized for procedural tasks performs significantly better than heuristics leading to a mAcc boost from $12.7 \rightarrow 24.1\%$ (row 2 and 3, $l = 4$, Tab. 1). With our novel decomposition and pretraining objective, VLaMP outperforms these baselines with a further bump up from $24.1 \rightarrow 31.7\%$ (row 3 and 4).

Steady gains in short & long horizon predictions. Comparing columns corresponding to $l = 1$ and $l = 4$, the performance (as one might expect) decreases, across all baselines and tasks. However, zooming in on COIN results, we observe consistent gains of VLaMP over DDN. Particularly, a relative improvement of 54% ($29.3 \rightarrow 45.2$) for $l = 1$ (row 7 and 8, Tab. 1) and a relative improvement of 68% ($21.0 \rightarrow 35.2$) in mAcc.

[3] Since we evaluate for three and four next steps on our datasets, we use the maximum required length and set $l = 4$.

[4] We detail these adaptations in Appendix B.1.

[5] Additional details are deferred to Appendix B.

Data	Model	$l = 1$	$l = 3$			$l = 4$		
		{n/m}Acc	SR	mAcc	mIOU	SR	mAcc	mIOU
Cross-task	Random	0.9	0.0	0.9	1.5	0.0	0.9	1.9
	Rand.(w/ G)	13.2	0.3	13.4	23.6	0.0	12.7	27.8
	DDN [4]	33.4	6.8	25.8	35.2	3.6	24.1	37.0
	VLaMP (ours)	**50.6**	**10.3**	**35.3**	**44.0**	**4.4**	**31.7**	**43.4**
COIN	Random	0.1	0.0	0.1	0.2	0.0	0.1	0.2
	Rand.(w/ G)	24.5	1.7	21.4	42.7	0.3	20.1	47.7
	DDN [4]	29.3	10.1	22.3	32.2	7.0	21.0	37.3
	VLaMP (ours)	**45.2**	**18.3**	**39.2**	**56.6**	**9.0**	**35.2**	**54.2**

Table 1: **Performance on different datasets and horizons.** The mean of various metrics (Sec. 2) obtained using 5 runs with different random seeds (std. errors are provide in Appendix I). Note that the action and observation history are the output of the separately finetuned video-action segmentation model and hence are noisy compared to the ground truth history.

Privileged random baseline comes close to DDN on easy metrics. Interestingly, we observe the performance of *Random w/ goal* comes close to DDN, when evaluated for lenient metrics like mAcc and mIOU (in COIN section of Tab. 1). Note, *Random w/ goal* enjoys the privileged access to (a much smaller) 'relevant actions set' for a given goal. On an average, the relevant or feasible action set is smaller for COIN than CrossTask videos. This a priori access to feasible actions makes *Random w/ goal* a competitive baseline.

Goal-conditioning is crucial. A key difference between procedure planning and VPA is goal conditioning is provided to the neural policies. In Tab. 5 (rows 1 and 2), we measure the effect of providing the goal as a *textual description* for last-observation-only models. The only difference is goal prompt G, which increases mAcc performance from $44.5 \rightarrow 53.1\%$ for $l = 1$ and $28.3 \rightarrow 34.7\%$ for $l = 3$.

5. Conclusion

Visual Planning for Assistance is a new and intuitive formulation to support planning from raw visual observations for assisting humans in day-to-day activities. We benchmark VPA using standard prior work and a new multi-modal sequence modeling formation of VLaMP. The novel decomposition of a VLaMP policy into video action segmentation and forecasting leads to several efficiency and modeling benefits. Particularly, this allows to leverage pre-trained LM that leads to significant performance gains. Alternative decompositions and self-supervised pre-training objectives for PTLMs are interesting ways forward for VPA.

References

[1] Jing Bi, Jiebo Luo, and Chenliang Xu. Procedure planning in instructional videos via contextual modeling and model-

based policy learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15611–15620, 2021. 4, 12

[2] Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *6th Annual Conference on Robot Learning*, 2022. 1

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *NeurIPS*, 2020. 6, 7

[4] Chien-Yi Chang, De-An Huang, Danfei Xu, Ehsan Adeli, Li Fei-Fei, and Juan Carlos Niebles. Procedure planning in instructional videos. In *European Conference on Computer Vision*, pages 334–350. Springer, 2020. 4, 6, 9, 12

[5] Kristen Grauman, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, Hao Jiang, Miao Liu, Xingyu Liu, et al. Ego4d: Around the world in 3,000 hours of egocentric video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18995–19012, 2022. 1, 9

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 6, 8

[7] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022. 1, 6, 7

[8] Anastasis Kratsios, Behnoosh Zamanlooy, Tianlin Liu, and Ivan Dokmanić. Universal Approximation Under Constraints is Possible with Transformers, Feb. 2022. arXiv:2110.03303 [cs, math]. 1

[9] Esteve Valls Mascaro, Hyemin Ahn, and Dongheui Lee. Intention-conditioned long-term human egocentric action anticipation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 6048–6057, January 2023. 9

[10] Fabian Paischer, Thomas Adler, Vihang Patil, Angela Bitto-Nemling, Markus Holzleitner, Sebastian Lehner, Hamid Eghbal-Zadeh, and Sepp Hochreiter. History compression via language models in reinforcement learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 17156–17185. PMLR, 17–23 Jul 2022. 1

[11] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019. 3

[12] Jiankai Sun, De-An Huang, Bo Lu, Yun-Hui Liu, Bolei Zhou, and Animesh Garg. Plate: Visually-grounded planning with transformers in procedural tasks. *IEEE Robotics and Automation Letters*, 7(2):4924–4930, 2022. 4, 8

[13] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1207–1216, 2019. 2, 3

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017. 8

[15] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020. 3

[16] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 318–335, Cham, 2018. Springer International Publishing. 3, 6

[17] Hu Xu, Gargi Ghosh, Po-Yao Huang, Dmytro Okhonko, Armen Aghajanyan, Florian Metze, Luke Zettlemoyer, and Christoph Feichtenhofer. Videoclip: Contrastive pre-training for zero-shot video-text understanding. *arXiv preprint arXiv:2109.14084*, 2021. 3, 6

[18] Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. Are Transformers universal approximators of sequence-to-sequence functions? Feb. 2022. 1

[19] He Zhao, Isma Hadji, Nikita Dvornik, Konstantinos G Derpanis, Richard P Wildes, and Allan D Jepson. P3iv: Probabilistic procedure planning from instructional videos with weak supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2938–2948, 2022. 4, 8

[20] Dimitri Zhukov, Jean-Baptiste Alayrac, Ramazan Gokberk Cinbis, David Fouhey, Ivan Laptev, and Josef Sivic. Cross-task weakly supervised learning from instructional videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3537–3545, 2019. 2, 3

Appendix – Pretrained Language Models as Visual Planners for Human Assistance

We structure the supplementary material as follows:

A. Necessary details and clarifications that we couldn't include due to space in the main paper.
B. Implementation details of DDN [4]. Two new baselines – GPT3-based [3] language-only method and another based on most-probable actions.
??. Step-by-step algorithm for inference, for reproducibility and technical details.
D. Optimization, hardware, and training details associated with training VLaMP.
E. Comparisons to Ego4D's LTA benchmark.
I. Expanded empirical results, benchmarking the above additional baselines, and deep-dive into error analysis.

We will opensource our model code and weights, as well as the splits of the dataset to enable further research on VPA.

A. Clarifications

Due to space, some of the explanations in the main paper might be insufficient or unclear in a first read. Hence, we pre-emptively clarify some of these nuances in this section.

- VLaMP uses a randomly initialised forecasting model (L790) $\rightarrow$ This refers to VLaMP when its forecasting module is trained with random initialization, i.e., the transformer of the same architecture trained from scratch instead of LM pre-training.
- Per-second accuracy of video-action segmentation (L798-799) $\rightarrow$ The finetuned VideoCLIP [17] for video-action segmentation outputs an action label for each second of a video. We report this accuracy following the convention in the original VideoCLIP paper, averaged over all videos in the test set.
- (L615-617) A more accurate representation should include a the log-sum-exp operator. The correct expression is

$$L_{\text{act}}(\hat{h}_j) = h_j \cdot \hat{h}_j - \log \sum_{p=1}^{V_{\text{LM}}} \exp\left(h_p \cdot \hat{h}_j\right).$$

B. DDN and Additional Baselines

In this section we include more information about baselines that we benchmark on VPA in experiments (Sec. 4). *First*, we include necessary details of reproducing the DDN [4] and how we keep it consistent and fair to the proposed VLaMP. *Second*, we provide two additional baselines – a heuristic baseline, which leverages the structure of our goal-oriented activities for generating plans and a prompt-based baseline using a large LM. *Finally*, we briefly discuss prior procedural planning methods, which we choose not to compare with.

Data	Model	$l=1$ {n/m}Acc	$l=3$ SR	$l=3$ mAcc	$l=3$ mIOU
COIN	GPT-3 [7]	12.1	0.5	14.3	22.3
	VLaMP (ours)	**67.2**	**25.5**	**51.6**	**59.1**

Table 2: **Performance of VLaMP vs. GPT-3 prompting.** Inspired from [7], we sub-sample 310 videos from the test set of COIN to test a prompting-based, language-only baseline using GPT-3 [3], and compare it with VLaMP on the same videos. Such an application of prompting with GPT-3 does not perform well on VPA.

B.1. DDN [4]

Technical Background. Chang *et al*. [4] proposed Dual Dynamics Network (DDN) for procedural planning. The objective is to learn a latent space representation of observations and actions in addition to a dynamics and conjugate dynamics model that operate over this latent space. The latent representations and recurrent RNN-based dynamics model are learned together by minimizing a joint loss over predicted observations and actions. Such dynamics modeling in latent space is similar in spirit to the forecasting module in VLaMP (Eq. (2)).

Implementation. As shown in Fig. 5, we instantiate DDN for VPA by using an LSTM-based [6] f_{seq} in the forecasting module, which operates over the observation representations obtained using the same observation encoder f_{obs} consisting of pretrained S3D [16] and a mapper as VLaMP and action representations from an embedding layer-based action encoder f_{act}. Unlike VLaMP, where the mapper aims to project the visual observation representations into the input space of the pretrained LM, the mapper in DDN only provides trainable parameters to finetune the frozen S3D representations for downstream dynamics model. Both f_{seq} and f_{act} are initialized with random weights. Just as VLaMP, DDN is trained using cross-entropy loss for predicted actions and mean-squared error for predicted observation representations to jointly learn f_{obs}, f_{act}, and the sequence model. At inference, the model is unrolled autoregressively (with beam search as shown in Algo. 1), for prediction of both action and observation representations. These design choices are consistent with VLaMP.

B.2. Additional Baseline: GPT-3 Planner (Tab. 2)

Following Huang *et al*. [7], where the authors use a LLM as zero-shot planners, we too also experiment with prompting a frozen pretrained large language model (GPT-3) for VPA. Specifically, a goal prompt and the current history

Algorithm 1: Inference for VLaMP and baselines

Data: encoded representations for history $H_k = (h_1, \ldots, h_n)$, beam size B
Result: plan for next l steps $\hat{\mathcal{T}} = a_{k+1}, \cdots, a_{k+l}$

$\mathcal{H}_0 \leftarrow \{H_k\}$; // Initialize the set of encoded trajectories with the history
for $i = 1, \ldots, l$; // predict actions for l steps
do
  $\mathcal{H}_i \leftarrow \{H \diamond f_{\text{enc}}(a) \mid H \in \mathcal{H}_{i-1}, a \in \mathcal{A}\}$; // All single action extension at i-th step
  $\Phi_i \leftarrow \{\phi(H) \mid H \in \mathcal{H}_i\}$; // score each trajectory
  $\tilde{\mathcal{H}}_i, \tilde{\Phi}_i \leftarrow \texttt{top}(B, \texttt{sorted}(\mathcal{H}_i, \Phi_i))$; // Keep B highest scoring trajectories
  $\tilde{\mathcal{H}}_i^0 \leftarrow \tilde{\mathcal{H}}_i$;
  for $u = 1, \ldots, \delta$; // Predict δ observation representations autoregressively
  do
    $\tilde{\mathcal{H}}_i^u \leftarrow \{H \diamond f_{\text{seq}}(H) \mid H \in \tilde{\mathcal{H}}_i^{u-1}\}$
  end
  $\mathcal{H}_i \leftarrow \tilde{\mathcal{H}}_i^{\delta}$;
end
$\hat{\mathcal{T}} \leftarrow \texttt{readout}(l, \texttt{top}(1, \texttt{sorted}(\tilde{\mathcal{H}}_l, \tilde{\Phi}_l)))$; // Read out last l actions from the top scoring beam

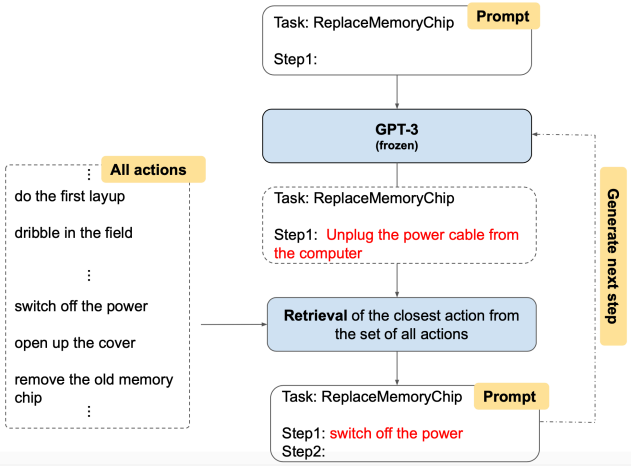
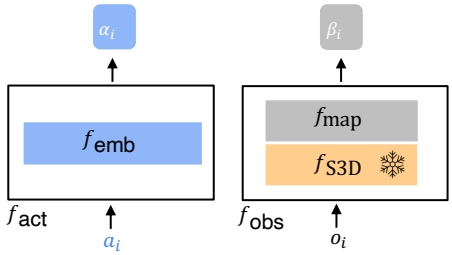
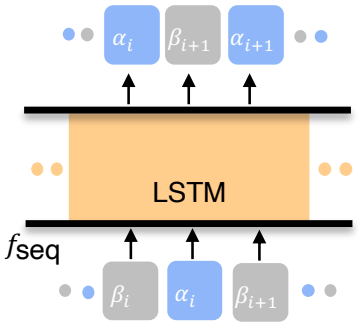


Figure 4: **GPT-3 as a planner based on [7].** A *language-only* baseline using GPT-3 on COIN. GPT-3 is prompted autoregressively to generate next action based on the goal and the history of actions taken for the goal.



(a) Encoders



(b) Sequence Model

Figure 5: **DDN Implementation.** The method utilizes LSTM in the forecasting module for VPA. Consistent to and analogous of VLaMP's Fig. 3 in the main paper.

of previously predicted actions (if available) are given as a prompt to the GPT-3 model [3]. Then the next actions for the given goal are generated autoregressively, consistent with other baselines like VLaMP. As can be seen in Fig. 4, this model has 2 stages: 1) next-action generation, and 2) action retrieval. In the next-action generation stage, the model is given the prompt and generates the next action. In the action retrieval stage, the generated next action is compared to all possible actions and the action that has the closest similarity to the generated action, is chosen and placed in the prompt. This step is required since we evaluate the generated plans by comparing with ground truth actions for the goal, where actions belong to a closed set as described in Sec. **??**. We use *text-davinci-003* backend for generation, and *text-embedding-ada-002* for embedding, which is used in combination with cosine similarity to retrieve the closest action. We perform our generation step zero-shot without giving any examples, as GPT3 can already follow the given prompt template, and we only generate one action at the time – in an autoregressive manner.

We sub-sampled the COIN test set, and compared VLaMP with the aforementioned GPT3-based model. In order to make the dataset suitable for language-only instantiation of VPA, we removed videos that had less than four

unique actions. We then evaluate both models on this subset of COIN. As can be seen in Tab. 2, VLaMP surpasses the GPT3-based baseline on all measures.

B.3. Additional Baseline: 'Most Probable Action'

We already benchmark two intuitive, heuristic baselines as discussed in Sec. **??** and Tab. 1 of the main paper, particularly, denoted as *random* and *random w/ goal*.

Here, we introduce an additional, heuristic baseline called 'most probable action'. The key insight here is that procedural activities are *highly structured*, *i.e.*, certain actions occur together or occur in a certain order. We bake this into a new baseline that captures this structure through the probability distribution of the next action given the current action.

To this end, the intuitive *most probable action* baseline picks the most probable next action $a_j|a_i$ from the action set $\mathcal{A}$. Akin to random w/ goal baseline, we also evaluate a goal-conditioned most probable action baseline, that uses a goal-specific set of actions $\mathcal{A}_G \subset \mathcal{A}$ during sampling. Since these most probable baselines, provide a probability distribution over the actions, we can employ beam search (for fairness, with the same beam size same as VLaMP) and pick the highest scoring plan. The results of this are included in Tab. 3 (an expanded version of Tab. 1 from the main paper).

B.4. On Porting More Baselines

Next, we briefly include some procedural planning approaches and reasons why they cannot be directly leveraged for rigorous and fair evaluation. Wherever possible, we include our best attempts to compare with them.

PlaTe [12]: This is similar to DDN, albeit with a Transformer [14] as the sequence model instead of an LSTM [6]. However, unlike DDN (and VLaMP), PlaTe uses separate Transformer-based models for state and action prediction. We adopt an approach that allowed us to tap into this while being consistent and fair in evaluation. Therefore, instead of directly adapting PlaTe for VPA as we did with DDN, we provide an ablation on VLaMP, which uses a Transformer trained from scratch as the sequence model (row **R** in Tab. 5).

P3IV [19]: This employs a significantly different modeling framework compared to DDN and PlaTe. Specifically, P3IV leverages a memory-augmented transformer as the sequence model and a probabilistic generative model to capture the noise and variability in predicted sequences. The authors report significant performance gain on the task of procedure planning, over DDN and PlaTe. However, P3IV relies on the visual observation of the goal already completed, even at inference time. This is necessary to condition their generative model towards encoding multiple plans from start to goal. Since P3IV needs the observations of goal completed, it is incompatible to the motivation and the very premise of VPA.

C. Inference for VPA (Alg. 1)

Before proceeding further, we pause and introduce additional notation for the sequence model, which will make the subsequent explanation for training and inference easier to follow. As shown in Figure 7, we alternatively denote the sequence of representations $(\beta_1, \alpha_1, \ldots, \beta_k, \alpha_k)$ by $H_k = (h_1, \ldots, h_n)$, with $n = k\delta + \sum_{i=1}^{k} r_i$. A binary mask $M = (m_1, \ldots, m_n)$, where m_i is 1 if the corresponding representation is for an action and 0 otherwise, can help obtain necessary action or visual observations. With this notation, given first j representations denoted as $h_{1:j}$, one step of the sequence model produces the representation for $j+1$, i.e., $f_{\text{seq}}(h_{1:j}) = \hat{h}_{j+1}$.

Now we explain the inference procedure for VLaMP briefly. Recall that we use $1:n$ to denote a sequence of n representations (i.e, $h_{1:n} = (h_1, \ldots, h_n)$). Additionally, we denote the concatenation operator over two representation sequences by "$\diamond$". With this notation at hand, we define the *score* of an action $a \in \mathcal{A}$ for following history $h_{1:n}$ as

$$\phi(h_{1:n} \diamond f_{\text{act}}(a)) = \sum_{j=1}^{r_a} a^j \cdot f_{\text{seq}}(h_{1:n} \diamond a^{1:j}), \qquad (5)$$

where $\cdot$ is the vector dot product, and $f_{\text{act}}(a) = \alpha^{1:r_a} = (\alpha^1, \ldots, \alpha^{r_a})$ is the sequence of encoded representations for action a. In other words, this *score* is the sum of unnormalized log-probability under the sequence model using the standard softmax distribution. We use this scoring function with to perform beam search.

C.1. Details of inference

In order to predict a sequence of next actions, we run the sequence model, autoregressively to predict both the action and observation tokens, with beam search on the action sequence. The inference algorithm is detailed in Algo. 1. We first encode the history into a sequence of representations H_k as described in Sec. 3.3, and initialize our set of encoder trajectories $\mathcal{H}_0$ using this single representation trajectory (line 1 in Alg. 1). Then we start the inference procedure that runs for l steps (line 2). At each step i we first infer the next action and then also predict the representations for the observation that follows it. To do the former, each representation trajectory in $\mathcal{H}_i$ is extended with the representations of each action in the action set $\mathcal{A}$ (line 4). At this point, if, for instance, $\mathcal{H}_{i-1}$ had n trajectories, then after line 4, $\mathcal{H}_i$ will have $n \times |\mathcal{A}|$ trajectories. This is a temporary blow-up– at line 6, we score all $n \times |\mathcal{A}|$ trajectories and keep only top B trajectories. Here, to balance diversity, we keep no more than b trajectories with exactly same history. The parameter b is usually referred to as *per node beam size*. Once we have

Dataset	Method	$l = 1$	$l = 3$			$l = 4$		
		nAcc	SR	mAcc	mIOU	SR	mAcc	mIOU
CrossTask	Random	0.9 ± 0.0	0.0 ± 0.0	0.9 ± 0.0	1.5 ± 0.0	0.0 ± 0.0	0.9 ± 0.0	1.9 ± 0.0
	Random w/ goal	13.2 ± 0.2	0.3 ± 0.0	13.4 ± 0.0	23.6 ± 0.1	0.0 ± 0.0	12.7 ± 0.0	27.8 ± 0.1
	Most probable	10.4 ± 0.0	1.7 ± 0.0	6.1 ± 0.0	9.9 ± 0.0	1.3 ± 0.0	5.5 ± 0.0	13.9 ± 0.0
	Most probable w/ goal	12.4 ± 0.0	2.4 ± 0.0	8.9 ± 0.0	15.5 ± 0.0	1.5 ± 0.0	7.9 ± 0.0	20.5 ± 0.0
	DDN [4]	33.4 ± 0.5	6.8 ± 0.3	25.8 ± 0.5	35.2 ± 0.6	3.6 ± 0.2	24.1 ± 0.4	37.0 ± 0.4
	VLaMP *(ours)*	**50.6 ± 1.4**	**10.3 ± 0.4**	**35.3 ± 1.1**	**44.0 ± 1.0**	**4.4 ± 0.2**	**31.7 ± 1.0**	**43.4 ± 0.9**
COIN	Random	0.1 ± 0.0	0.0 ± 0.0	0.1 ± 0.0	0.2 ± 0.0	0.0 ± 0.0	0.1 ± 0.0	0.2 ± 0.0
	Random w/ goal	24.5 ± 0.2	1.7 ± 0.0	21.4 ± 0.1	42.7 ± 0.1	0.3 ± 0.0	20.1 ± 0.1	47.7 ± 0.1
	Most probable	0.7 ± 0.0	1.6 ± 0.0	4.3 ± 0.0	6.8 ± 0.0	1.6 ± 0.0	8.2 ± 0.0	15.3 ± 0.0
	Most probable w/ goal	23.9 ± 0.0	10.9 ± 0.0	18.0 ± 0.0	24.9 ± 0.0	9.1 ± 0.0	16.3 ± 0.0	32.2 ± 0.0
	DDN [4]	29.3 ± 0.3	10.1 ± 0.4	22.3 ± 0.4	32.2 ± 0.6	7.0 ± 0.3	21.0 ± 0.4	37.3 ± 0.3
	VLaMP *(ours)*	**45.2 ± 0.8**	**18.3 ± 0.1**	**39.2 ± 0.3**	**56.6 ± 0.5**	**9.0 ± 0.3**	**35.2 ± 0.2**	**54.2 ± 0.5**

Table 3: **Expanded version of Tab. 1.** The mean ± std. error of mean for various planning metrics obtained using 5 runs with different random seed are shown for VLaMP and various baselines. Note that the action history and observations are provided using the output of the action segmentation model and hence are noisy compared to the ground truth history.

	beam size (B)	per node beam size (b)	GPU	GPU memory	Num GPUs (inference)	Num GPUs (training)	Avg. time (training)	Avg. time (inference)	batch size (training)
CrossTask	10	3	NVIDIA A100	80GB	1 GPU/model	1 GPU/model	2 s/batch	7.4 s/example	4
COIN	3	3	NVIDIA A100	80GB	3 GPUs/model	1 GPU/model	2 s/batch	6.1 s/example	4

Table 4: Hyperparameters and compute information for VLaMP.

B trajectories in $\mathcal{H}_i$, we auto-regressively predict the next δ tokens corresponding to the next observation, thus completing one out of the l steps of inference. This process it repeated l times to generate a plan consisting of l actions. We make this process efficient by storing the hidden state of the transformer and limiting the forward pass only on the new representations at each step. This is a common practice for transformer based models in NLP. Due to beam search, the inference process is slower than training as shown in Tab. 4.

D. VLaMP Training (Tab. 4)

Unlike inference where a video with K steps results into $K - 4$ examples, during training, like with language model pre-training, we use a single forward pass to compute loss for all tokens. Moreover, inference also uses beam search making it more memory intensive. Thus, the training is much faster and cheaper as compared to the inference. The details of the compute used for each training and inference run is shown in Tab. 4.

E. Comparison to Ego4D LTA Benchmark

In prior work, Ego4D's Long-term Action Anticipation [5] benchmark task is likely the most relevant to VPA. Hence, we dedicate a discussion of similarities and contrasts. We hope this helps the reader accurately place these two tasks in our community's diverse research goals and directions.

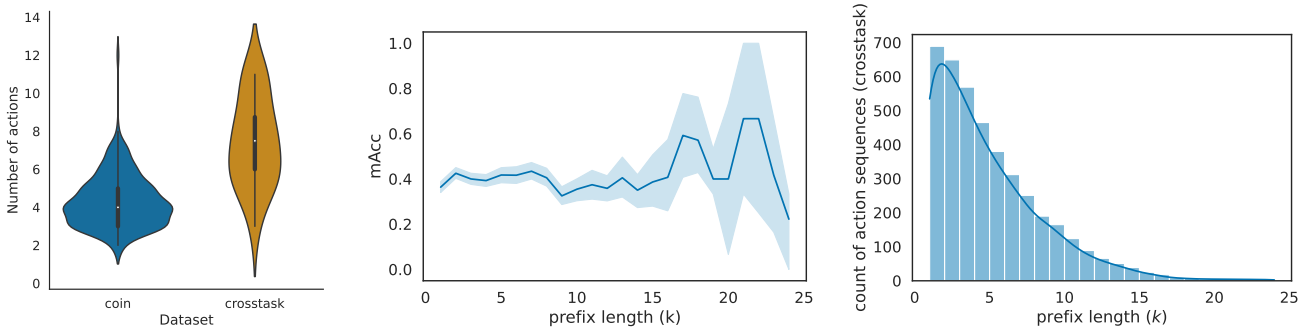
Consistent to VPA, LTA also focuses on predicting a sequence of future actions given prior visual context for free-form human interaction. Unlike LTA, VPA specifically entails *goal-oriented* activities and indeed a natural language goal prompt is of key importance to the definition of VPA. So while the forecasting suite in Ego4D aspires to understand human motion, we are instead keen to create assistive agents that can interact and assist humans in their tasks.

Since LTA does not allow access or model the user's goal, recent approaches for LTA including the winning model for Ego4D 2022 LTA challenge – ICVAE [9] have to go via an additional step of inferring the intention of the user. This provides more impetus to our goal-conditioned and human-assitive design choice and motivation for VPA. This is empirically backed as well, as we show in ablation (row 1 in Tab. 5) – goal-conditioning is crucial for VPA.

I. Additional Quantitative Results[6]

Most probable action baseline. As shown in Table 3, the performance of our intuitive heuristic baselines – *most probable action w/ goal*, and *Random w/ goal* (*i.e.* the baselines with actions restricted to the set of actions seen with the

[6]We number this section as I, to be consistent with the main paper references.



(a) $\mathcal{A}_G$ across COIN and CrossTask (b) Mean accuracy (mAcc) averaged across goals (c) Number of samples for each prefix length (length of history)

Figure 6: **Zooming into the tails.** (a) Average size of goal-specific action set $\mathcal{A}_G$ across COIN and CrossTask datasets. COIN has a relatively smaller mean than CrossTask, which reduces the difficulty of VPA on COIN. (b) Mean accuracy (mAcc) *vs.* the history length k of various goals from CrossTask. Interestingly, plan generation for goals with longer history is difficult and prone to higher errors as reflected in the mAcc (reasoning included in Sec. I). (c) Longer sequences are also less frequent in the dataset. This contributes to high variance in performance for such sequences.

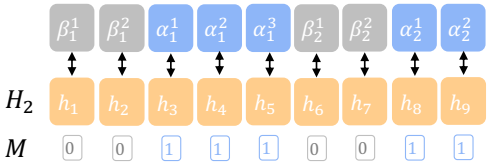


Figure 7: **Tokenized Sequence with Masks.** The encoded sequence of representations for $k = 2$ segments, denoted alternatively using modality agnostic notation of H_2 and mask M for next token prediction training.

corresponding goal) is quite high for COIN dataset. We find that this is due to the relatively small cardinality of the action set for goals in COIN, i.e. the average size of $\mathcal{A}_G$ for different Gs.

Action distribution analysis. We dig deeper into the above finding in Fig. 6(a). Particularly, we plot the distribution of number of actions $|\mathcal{A}_G|$ w.r.t G and find them to be quite different in COIN vs. CrossTask. Here, $|\mathcal{A}_G| \in \mathcal{A}$ represents the set of goal-specific actions from the larger set of actions $\mathcal{A}$ for each dataset. Specifically, the average size of $\mathcal{A}_G$ is 4.3 and 7.3, respectively for COIN and CrossTask, reducing the difficulty of VPA in COIN. Also, notice the long-tails in the distribution of CrossTask, making it even more challenging.

Zooming into the tails and higher errors. In Fig. 6(b), we plot the number of steps in the history (k) *vs.* the mean accuracy (mAcc), averaged across all goals in CrossTask on VPA. We find that plan generation with *longer history leads to higher errors* as well as higher variance in performance. We believe this trend emerges due to *two reasons*.

First, the presence of repetitive steps in certain goals is high in longer history. Moreover, we find that longer the history the wider is the space of possible plans (intuitively, multiple modes exist in the plan distribution landscape), which lead to higher variance. We illustrate this in Fig. 8 with a qualitative result, for the example goal of 'making kimchi fried rice', the action 'stir mixture' repeatedly occurs between various actions involving the addition of ingredients like onion, kimchi, rice, etc. However, the number of times *stir mixture* occurs varies sporadically. For instance, for the ground truth plan in the first example in Fig. 8, the 'stir mixture' is missing between 'add onion' and 'add kimchi', but occurs twice after 'add kimchi', before adding other ingredients. Due to this sporadic variability, the predicted plan gets IoU of 75% on this example, but mAcc and SR of 25% and 0, respectively. Another common source of errors is repetition of sub-sequences of actions depending on the visual signal in the ground truth. Specifically, as seen in the second example in Fig. 8, which shows an action trajectory for the goal of 'jack up a car', the sub-sequence ('raise jack', 'lower jack'), is repeated three times. In this example, the repetition is due to overshooting the target height of the raised car. However, for a planning model, that only sees the visual input till $k = 3$ or time t_k, it is not possible to guess whether the car will overshoot (undershoot, respectively) the target height after the next application of 'raise jack' ('lower jack', respectively).

Second, as analysed in Fig. 6(c), *longer trajectories are exponentially less frequent* in the dataset – forming the tail of the data distribution of action sequences in the dataset. This also contributes to high variance in performance for such sequences.

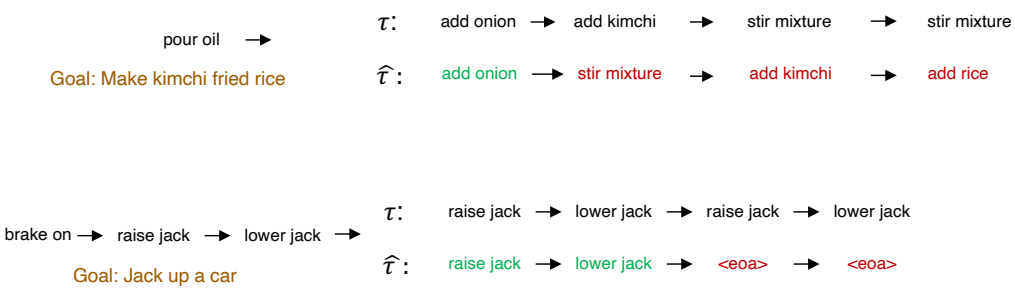


Figure 8: **Qualitative Error Analysis for VPA.** Ground truth plan $\mathcal{T}$ and the predicted plan $\hat{\mathcal{T}}$ by VLaMP for the goal prompt of "making kimchi fried rice" (*top*) and " Jack up the car" (*bottom*). Errors made by VLaMP can be attributed to *repetitions in actions*. Details are included in Sec. I. Briefly, (1) uncertainty in the number of times actions are repeated and (2) existence of equivalent plans for achieving the same goal, are contribute heavily to the errors for VPA. In the top, note the action 'stir mixture' is repeated consecutively in the ground truth, but the model predicts it only once. Moreover, both the ground truth and the predicted plans have correct steps for adding kimchi and onion but their *order is different*. Similar repetitions result into errors for the goal of jacking up the car.

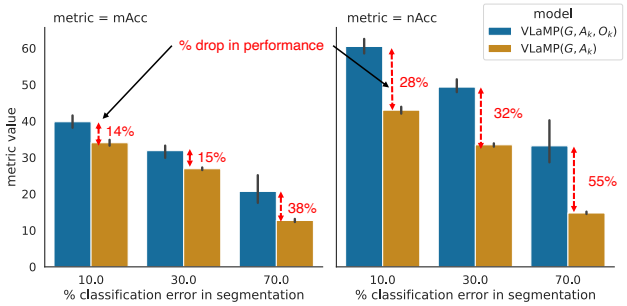


Figure 9: **Effect of segmentation errors.** The figure zooms in on two metrics mAcc and nAcc from Figure 10. As the classification error in segmentation, which is shown along the x-axis increases, the performance gap between the model with access to observation history VLaMP (G, A_k, O_k) and that with access only to the action history VLaMP (G, A_k) increases.

Segmentation errors. As we note in Sec. **??** and Fig. 10 of the main paper, segmentation errors are detrimental for VPA. As the mis-classification error in the segmentation model increases, the difference in the performance of VLaMP (G, A_k, O_k), i.e. the model with access to observation history, and VLaMP (G, A_k), the model working only on the action history, increases. A detailed version of Fig. 10 is included in Fig. 9 with a focus on mean accuracy (mAcc) and next-step accuracy (nAcc)[7]. Moreover, since the observation history has higher influence on predicting the immediate next action as discussed in Sec. **??**, the performance drop due to segmentation classification error is higher in nAcc as compared to mAcc.

[7]Refer to the definitions of metrics in Sec. **??**

	G	A_k	O_k	$l=3$ SR	$l=3$ mAcc	$l=3$ mIOU	$l=1$ nAcc
1	✗	✗	o_k	6.8 ± 0.3	28.3 ± 1.9	34.8 ± 2.0	44.5 ± 3.8
2	✓	✗	o_k	8.9 ± 0.2	34.7 ± 0.7	41.6 ± 0.8	53.1 ± 2.0
3	✓	✓	✗	14.9 ± 0.3	37.8 ± 0.4	50.8 ± 0.6	48.0 ± 0.2
4	✓	✓	O_k	15.2 ± 0.3	43.5 ± 0.8	51.4 ± 0.9	64.8 ± 0.9
R	✓	✓	O_k	10.7 ± 0.2	36.5 ± 0.7	41.7 ± 0.6	61.4 ± 1.8

Table 5: **Role of different inputs and LM pre-training in VLaMP.** Here G, A_k, O_k specify the inputs provided to VLaMP during training and inference, in terms of goal, action history and observation history respectively. o_k refers to the use of most recent observation from the history instead of the full observation history. The mean ± std. error over 5 runs with different random seeds on CrossTask are shown. The row **R** corresponds to VLaMP trained with random initialization as compared to LM pretraining.

I.1. Ablations and Error Analysis

Utilizing head-on ablations, we evaluate how the action history and the visual history affect the plan generation for VPA. To remove confounding factors, in Tab. 5 we use the ground truth output for the segmentation module.

Action and observation history improve complementary planning metrics. As seen in all the rows with action history Tab. 5, the provision of action history increases difficult metrics such as SR. In comparing rows 2 and 3, we see that SR increases by about 67% (8.9 → 14.9), simply by using action history even without access to any past observation. However, the lack of observation history affects nAcc, which drops by 10% (53.1 → 48.0) when observation history is swapped by action history between rows 2

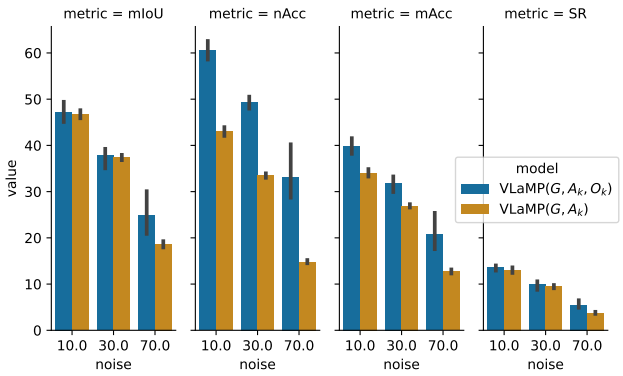


Figure 10: **Effect of segmentation errors.** Performance of VLaMP on CrossTask with classification noise (%) in the segmentation.

and 3.

Priors from the pre-trained LM improve performance. The row **R** of Tab. 5 shows the performance of VLaMP when its forecasting module is trained with random initialization, i.e., the transformer of the same architecture trained from scratch instead of LM pre-training. The performance in row **R** is thus much lower than that of VLaMP with pre-trained LM shown in Tab. 1, which highlights the importance of LM pre-training.

Segmentation errors are detrimental. The accuracy of finetuned VideoCLIP model for video-action segmentation is 80.2 and 68.7 when segmenting a video per second for CrossTask and COIN datasets respectively. The effect of such segmentation error in VLaMP's performance can be observed by comparing row 4 of Table 5, which uses *ground-truth* segments, with row 4 of Table 1 that uses a finetuned segmentation model – all metrics significantly drop due to the introduced segmentation error.

Visual history aids in planning when segmentation has errors. The error of the segmentation module may lead to mis-classification of actions leading to erroneous action history. In order to systematically study the effect of erroneous action history on VLaMP's performance in visual planning, decoupled from VideoCLIP's segmentation accuracy, we perform controlled experiments wherein we add noise in the ground truth segmentation. Specifically, we replace varying amounts of actions in the ground truth action history by random actions. We also compare two models –VLaMP(G, A_k, O_k), which uses both action and observation history, and VLaMP(A, A_k), which only uses action history. As the noise increases, the gap in the performance of the model with access to visual history and the one without, rises. This provides evidence for usefulness of observations for robustness against segmentation classification error.

Errors increase towards the tail of the activities. In

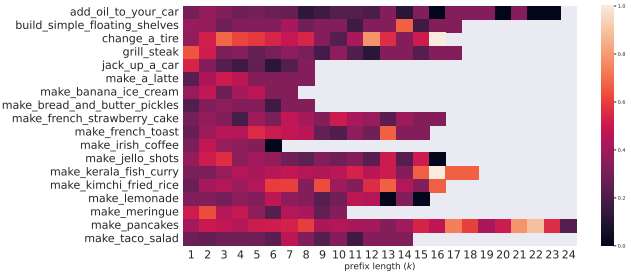


Figure 11: **Prediction in the tail of long activities is challenging.** mAcc for $l = 3$ w.r.t. the number of steps in history (k).

Fig. 11, we shows mAcc w.r.t. the number of steps k in the history. The performance drops towards the tail of the activities inspite of access to longer history. Furthermore, this drop is most significant for the activities that have many steps like make pancakes, add oil to car, etc. We hypothesize that such a drop is due to two reasons: high uncertainty about the completion of the activity and relatively less number of training examples in the long tail region of each activity. We will discuss this more in detail in Appendix I.

J. Metrics

The planning performance of a VPA model is measured by comparing the generated plan $\hat{\mathcal{T}} = (\hat{a}_{k+1}, \ldots, \hat{a}_{k+l})$ to the ground truth plan $\mathcal{T}$ for l actions in the future, given V_t and G. Here $\hat{a}_{k+i}$ denotes the prediction for the $k+i$-th step given history till k-th step. Consistent with community practices [4, 1], we use the following metrics, listed in decreasing order of strictness: *success rate* (SR), *mean accuracy* (mAcc), *mean intersection over union* (mIOU). Success rate requires an exact match between all actions and their sequence between $\hat{\mathcal{T}}$ and $\mathcal{T}$. Mean accuracy is the accuracy of the actions at each step. Unlike success rate, mean accuracy does not require a 100% match to ground truth. Instead it considers matching at each individual step. Lastly, mean intersection over union captures the cases where the model predicts the steps correctly, but fails to identify the correct order. Concretely,

$$\text{mIOU}_l = \frac{\left|\hat{a}_{\{k+1:k+l\}} \bigcap a_{\{k+1:k+l\}}\right|}{\left|\hat{a}_{\{k+1:k+l\}} \bigcup a_{\{k+1:k+l\}}\right|}, \tag{6}$$

$$\text{mAcc}_l = \frac{1}{l}\sum_{i=1}^{l} \mathbb{1}[\hat{a}_{k+i,} = a_{k+i}], \tag{7}$$

$$\text{SR}_l = \prod_{i=1}^{l} \mathbb{1}[\hat{a}_{k+i} = a_{k+i}], \tag{8}$$

where $\mathbb{1}[\cdot]$ is the identity function, which is 1 when the condition in its input is true, and 0, and $\hat{a}_{\{k+1:k+l\}}$ denotes the

set of l future actions in $\hat{\mathcal{T}}$, i.e., a sequence but disregarding the order. To complement the above metrics, we also measure the accuracy of predicting the next action *i.e*. nAcc, as defined in (9), where 'n' stands for next. Note that all metrics are averaged over the test set details of which are included in Sec. 4.

$$\text{nAcc} = \mathbb{1}[\hat{a}_{k+1} = a_{k+1}] \tag{9}$$

K. Segmentation Module

This module splits the untrimmed video history V_t into segment history S_k of multiple segments, each segment corresponding to an action. The segmentation is done using a video-action segmentation model in three steps: preprocessing, classification, and consolidation. In the preprocessing step, the raw video frames from V_t are bundled into fixed-length window clips c_i, each of length 1 second, to obtain $V_t = (c_1, \ldots, c_t)$. In the classification step, a video-action segmentation model is used to output the most probable action for each clip c_i, which can be denoted as $\tilde{A}_t = (\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_t)$. Finally, in the consolidation steps, we convert $\tilde{A}_t$ into a form that can be used by the forecasting module; this form consists of two sequences: **action history** A_k and **observation history** O_k. To this end, same actions in consecutive seconds in $\tilde{A}_t$ are consolidated to form the action history $A_k = (a_1, \ldots, a_k)$. As illustrated in Fig. 2, assuming t_i denotes the starting timestamp for a_i, we also extract video frames from $t_i - \delta/2$ to $t_i + \delta/2$ to obtain a *observation window* o_i corresponding to a_i, and consequently the full observation history $O_k = (o_1, \ldots, o_k)$. The resultant segment history is termed $S_k = ((o_1, a_1), \ldots, (o_k, a_k))$, summarized in Fig. 2.