Appendix – Pretrained Language Models as Visual Planners for Human Assistance

We structure the supplementary material as follows:

A. Implementation details of baselines: DDN [9], GPT3-based [8, 30] language-only method and most-probable actions.
B. Step-by-step algorithm for inference, for reproducibility and technical details.
C. Optimization, hardware, and training details associated with training VLaMP.
D. Comparisons to Ego4D's LTA benchmark.
E. Expanded empirical results, benchmarking the above additional baselines, and deep-dive into error analysis.

A. Baselines

In this section we include more information about baselines that we benchmark on VPA in experiments (Sec. 5). *First*, we include necessary details of reproducing the DDN [9] 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. As mentioned in the main paper, we also provide the std (standard error of mean) around the mean for various models in Tab. 5.

A.1. DDN [9]

Technical Background. Chang *et al*. [9] 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. (7)).

Implementation. As shown in Fig. 8, we instantiate DDN for VPA by using an LSTM-based [28] f_{seq} in the forecasting module, which operates over the observation representations obtained using the same observation encoder f_{obs} consisting of pretrained S3D [84] 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,

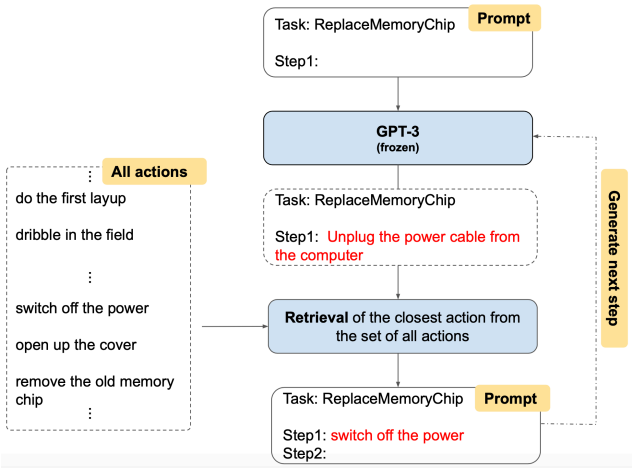


Figure 7: **GPT-3 as a planner based on [30].** A *language-only* baseline using GPT-3 on COIN. GPT-3 is prompted autoregressively to generate next action based on the goal and the history of actions taken for the goal.

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.

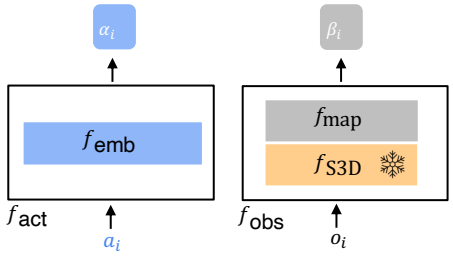
A.2. GPT-3 Planner

Following Huang *et al*. [30], 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 of previously predicted actions (if available) are given as a prompt to the GPT-3 model [8]. Then the next actions for the given goal are generated autoregressively, consistent with other baselines like VLaMP. As can be seen in Fig. 7, 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. 3.2. 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

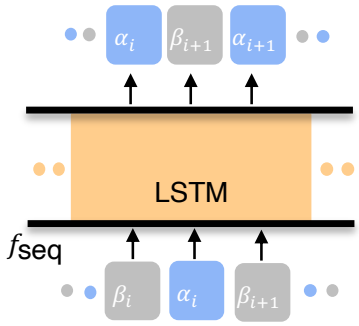
Algorithm 1: Inference for VLaMP and baselines

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



(a) Encoders



(b) Sequence Model

Figure 8: **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.

at the time – in an autoregressive manner.

A.3. 'Most Probable Action' baseline

In addition to the two intuitive, heuristic baselines, *random* and *random w/ goal*, we also use a stronger heuristic baseline called 'most probable action' baseline. Here we describe this baseline in detail. The idea is to leverage the fact that procedural activities are *highly structured*, *i.e.*, certain actions occur together or occur in a certain order. We bake this into a simple model with Markov assumption, that the probability distribution of the next action a_{k+1} given the current action a_k to predict future actions. 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.

A.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 [70]: This is similar to DDN, albeit with a Transformer [75] as the sequence model instead of an LSTM [28]. 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. 4).

P3IV [87]: 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

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
	DDN [9]	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
	DDN [9]	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 5: **Expanded version of Tab. 3.** The mean ± ste. (standard error of mean) for various planning metrics obtained using 5 runs with different random seed are shown for VLaMP and various baselines. Note: the models that don't change with random seeds are omitted. 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 6: Hyperparameters and compute information for VLaMP.

from start to goal. Since P3IV needs the observations of goal completed, it is incompatible to the motivation and the very premise of VPA.

B. Inference for VPA (Alg. 1)

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. 4.3, 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 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. 6.

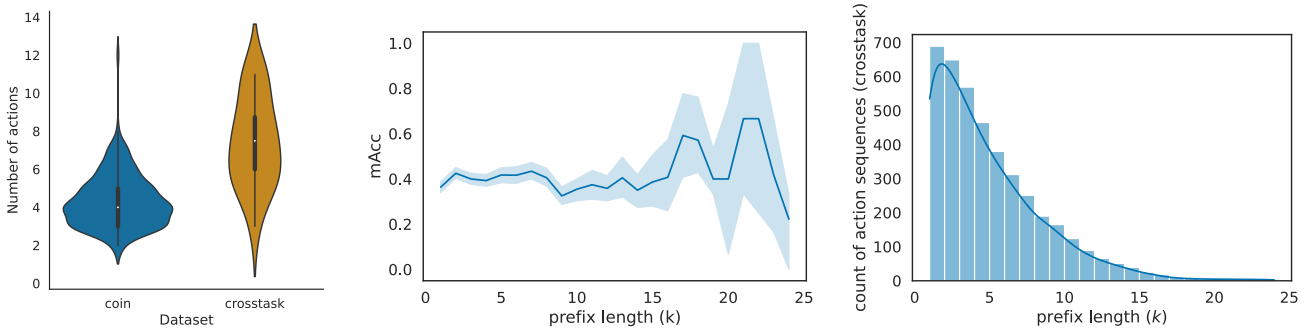
C. VLaMP Training (Tab. 6)

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. 6.

D. Comparison to Ego4D LTA Benchmark

In prior work, Ego4D's Long-term Action Anticipation [25] benchmark task is also highly 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



(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 9: **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. E). (c) Longer sequences are also less frequent in the dataset. This contributes to high variance in performance for such sequences.

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 [51] 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. 4) – goal-conditioning is crucial for VPA.

We also evaluated VLaMP on Ego4D's LTA benchmark. Both the VideoCLIP-based segmentation module and the forecasting module in VLaMP were finetuned on LTA's train split. Table 7 shows the performance of VLaMP on the validation split of LTA for $l = 20$ future actions, following the LTA task definition. The extremely long future horizon of prediction makes beam search impractical. Hence, we don't perform beam search with VLaMP for this experiment. VLaMP outperforms the best performing LTA baseline, which uses a visual encoder followed by a transformer-based aggregator [25]. While Ego4D used Slowfast encoder for the LTA baseline, we perform this experiment using S3D encoder for a fair comparison with VLaMP.

		ED@(l=20)		
Model	Encoder	Verb	Noun	Action
VLaMP (ours)	S3D	**0.73**	**0.772**	**0.932**
LTA Baseline [25]	S3D	0.745	0.779	0.941

Table 7: **VLaMP on Ego4D LTA's validation split.** Edit distance for verb, noun, action prediction for future 20 steps is shown (**lower** the better).

E. Additional Quantitative Results

Most probable action baseline. As shown in Table 3, the performance of the 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 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 as discussed below.

Action distribution analysis. We dig deeper into the above finding in Fig. 9(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. 9(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 (intu-

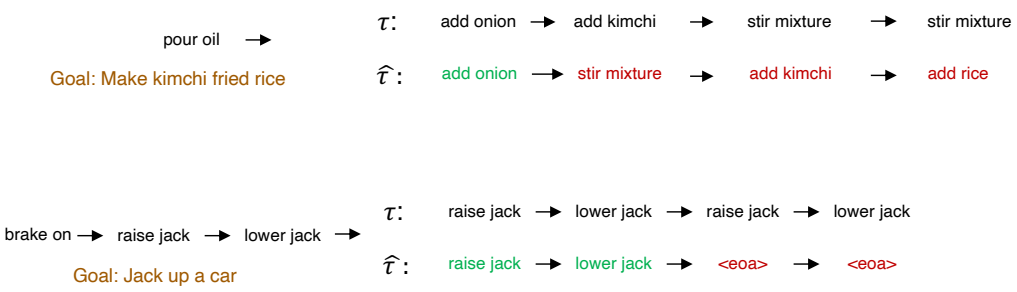


Figure 10: **Qualitative Error Analysis for VPA.** Ground truth plan $\mathcal{T}$ and the predicted plan $\hat{\mathcal{T}}$ by VLaMP for the goal prompt of "making kimchi fried rice" (*top*) and " Jack up the car" (*bottom*). Errors made by VLaMP can be attributed to *repetitions in actions*. Details are included in Sec. E. 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.

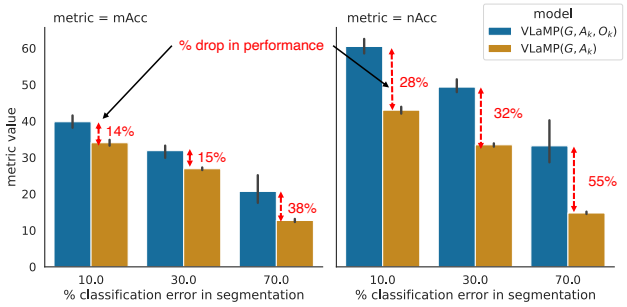


Figure 11: **Effect of segmentation errors.** The figure zooms in on two metrics mAcc and nAcc from Figure 5. As the classification error in segmentation, which is shown along the x-axis increases, the performance gap between the model with access to observation history VLaMP (G, A_k, O_k) and that with access only to the action history VLaMP (G, A_k) increases.

itively, multiple modes exist in the plan distribution landscape), which lead to higher variance. We illustrate this in Fig. 10 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. 10, 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. 10, 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. 9(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.

Segmentation errors. As we note in Sec. 5.3 and Fig. 5 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. 5 is included in Fig. 11 with a focus on mean accuracy (mAcc) and next-step accuracy (nAcc)[6]. Moreover, since the observation history has higher influence on predicting the immediate next action as discussed in Sec. 5.3, the performance drop due to segmentation classification error is higher in nAcc as compared to mAcc.

[6]Refer to the definitions of metrics in Sec. 3.2