ViterbiPlanNet: Injecting Procedural Knowledge via Differentiable Viterbi for Planning in Instructional Videos

Supplementary Material

Luigi Seminara[1†] Davide Moltisanti[2*] Antonino Furnari[1*]
[1]University of Catania [2]University of Bath
https://gigi-g.github.io/ViterbiPlanNet/

Table of Contents

Appendices **1**

1 . Method **1**
1.A . Problem Formulation (fn 1) 1
1.B . Viterbi Algorithm 2
1.C . Differentiable Viterbi (fn 5) 2
1.D . Visual Encoding (fn 5) 3
1.E . Training (fn 6) 3

2 . Experiments and Results **4**
2.A . Discussion on mIoU metric 4
2.B . Bootstrap Procedure for Statistical Significance (fn 6) 4
2.C . Ablations on CrossTask (fn 8) 5
2.D . Comparisons with the State of the Art (fn 8) 7
2.E . Comparison with MTID (fn 12) . . . 8

3 . Additional Studies **9**
3.A . Study on Rigidity of the Markov Assumption (fn 1) 9
3.B . Study on Dependence on PKG Quality (fn 2) 10
3.C . Dataset-Independent PKG (fn 3) . . 10
3.D . Intermediate Visual Observations (fn 4) 10
3.E . Planning in Egocentric Instructional Videos (fn 7) 11

4 . Further Experimental Details **11**
4.A . Hyperparameter Configuration 11
4.B . LLM and VLM Details (fn 12) . . . 11

*Equal advising.
†Work done while visiting the University of Bath.

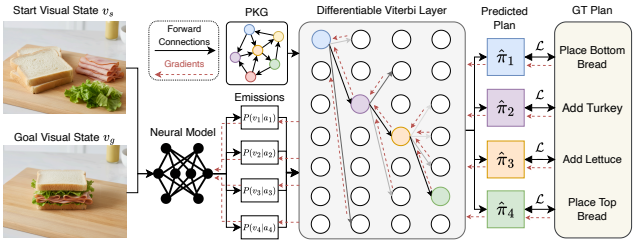

Figure 1. Given start and goal visual states, a neural model computes step-wise emissions. We propose a Differentiable Viterbi Layer that uses a Procedural Knowledge Graph (PKG) to decode emissions into a predicted plan. The layer allows gradients from the planning Loss ($\mathcal{L}$) to flow and train the neural model end-to-end, forcing it to learn structure-aware visual representations.

In this supplementary material we provide more details about our method (see Figure 1) and the experimental setup. We point to each footnote in the main paper with the following notation: (fn x) where x is the footnote number in the main paper.

1. Method

1.A. Problem Formulation (fn 1)

Marginalizing the initial latent action a_0. We consider a latent action sequence $a_{0:T} = (a_0, a_1, \ldots, a_T)$ and a corresponding sequence of observed visual states $v_{0:T} = (v_s = v_0, v_1, \ldots, v_T = v_g)$. Under the first–order Markov assumption, the model is factorized into independent transition and emission components. The joint distribution over latent actions and visual observations is therefore:

$$P(a_{0:T}, v_{0:T}) = P(a_0)\, P(v_0 \mid a_0) \prod_{t=1}^{T} P(a_t \mid a_{t-1})\, P(v_t \mid a_t). \quad (1)$$

Our objective is to infer the posterior distribution over the latent plan $\pi = a_{1:T}$ given the full sequence of visual observations. Using Bayes' rule and marginalizing the unobserved initial action a_0, we obtain:

$$P(a_{1:T} \mid v_{0:T}) = \frac{\sum_{a_0 \in \mathcal{K}} P(a_{0:T}, v_{0:T})}{P(v_{0:T})}$$
$$\propto \sum_{a_0 \in \mathcal{K}} P(a_0)\, P(v_0 \mid a_0) \prod_{t=1}^{T} P(a_t \mid a_{t-1})\, P(v_t \mid a_t), \tag{2}$$

where $\mathcal{K}$ denotes the discrete action set. Only the terms involving a_0 depend on the marginalization, and all other factors remain unaffected. It is therefore convenient to group the contributions of a_0 into an *effective prior* over the first action a_1. For any action class $K_j \in \mathcal{K}$, we thus have:

$$P(a_1 = K_j \mid v_0) \;\propto\; \sum_{a_0 \in \mathcal{K}} P(a_0)\, P(v_0 \mid a_0)\, P(a_1 = K_j \mid a_0). \tag{3}$$

However, in practice, the quantity $P(a_0)\, P(v_0 \mid a_0)$ cannot be estimated directly because the initial action is unobserved and the model lacks supervision for this term. Following common practice, we thus assume a *uniform prior* over the first action a_1. Under this assumption, all structural information at $t = 1$ is captured by the emission term $P(v_1 \mid a_1)$. Substituting this simplification into Eq. (2), the posterior becomes:

$$P(a_{1:T} \mid v_{0:T}) \;\propto\; \prod_{t=1}^{T} P(a_t \mid a_{t-1})\, P(v_t \mid a_t), \tag{4}$$

which is the quantity whose maximization yields the most probable latent plan.

1.B. Viterbi Algorithm

The Viterbi algorithm [11] is a dynamic programming method for computing the most likely sequence of latent states in a Hidden Markov Model (HMM). Given (1) a set of N discrete states $\mathcal{K} = \{K_1, \ldots, K_N\}$, (2) transition probabilities $P(a_t \mid a_{t-1})$, and (3) emission probabilities $P(v_t \mid a_t)$, the goal is to find the most probable sequence of hidden actions $a_{1:T}$ that explains the observations $v_{1:T}$.

Objective. The algorithm maximizes the posterior Eq. (4) as follows:

$$\pi^* = \underset{\pi = a_{1:T} \in \mathcal{K}^T}{\arg\max} \prod_{t=1}^{T} \underbrace{P(a_t|a_{t-1})}_{\text{Transition}} \underbrace{P(v_t|a_t)}_{\text{Emission}}. \tag{5}$$

Dynamic programming recursion. To efficiently compute (5), Viterbi stores: (1) *state scores* $\delta_t(j)$, the best score of any path ending in state K_j at time t; (2) *backpointers* $\psi_t(j)$, the most likely predecessor of K_j. Since no predecessor exists at $t = 1$, initial scores depend only on emissions:

$$\delta_1(j) = P(v_1 \mid a_1 = K_j), \qquad j = 1, \ldots, N. \tag{6}$$

For each time step $t > 1$ and each state K_j, Viterbi computes:

$$\delta_t(j) = P(v_t \mid a_t = K_j) \cdot \max_{i \in \{1,\ldots,N\}} \left[\delta_{t-1}(i)\, P(a_t = K_j \mid a_{t-1} = K_i)\right], \tag{7}$$

$$\psi_t(j) = \underset{i \in \{1,\ldots,N\}}{\arg\max} \left[\delta_{t-1}(i)\, P(a_t = K_j \mid a_{t-1} = K_i)\right]. \tag{8}$$

The $\max$ operator ensures that only the best predecessor state contributes to the score.

Backtracking. Once δ_T is computed, the final state is chosen as follows:

$$a_T^* = \arg\max_j \delta_T(j), \tag{9}$$

and the full optimal plan is reconstructed by tracing back the stored pointers:

$$a_t^* = \psi_{t+1}(a_{t+1}^*), \qquad t = T-1, \ldots, 1. \tag{10}$$

Interpretation. The Viterbi algorithm guarantees:

- **Optimality**: the returned sequence maximizes the posterior probability;
- **Efficiency**: complexity $O(TN^2)$ instead of exponential search $O(N^T)$;
- **Modularity**: transitions and emissions contribute separately, matching the probabilistic factorization used in Eq. (5).

Connection to our formulation. In our framework emissions are predicted from visual encodings, and transitions come from the Procedural Knowledge Graph (PKG). This correspondence makes Viterbi a natural decoding algorithm for procedural planning. However, its $\max$ and $\arg\max$ operations prevent end-to-end learning, motivating the introduction of our Differentiable Viterbi Layer (DVL), which replaces these operators with smooth relaxations following [7].

1.C. Differentiable Viterbi (fn 5)

Smooth max and soft argmax operators. Let $\mathbf{x} \in \mathbb{R}^N$ be a generic score vector, we define the following *smooth max* (log-sum-exp) and *soft argmax* (softmax) operations

which aim to extract from $\mathbf{x}$ a max-like value and a distribution over indexes corresponding to entries closer to the maximum value in a differentiable fashion:

$$\text{S-max}(\mathbf{x}) = \log\left(\sum_{k=1}^{N}\exp(x_k - m)\right) + m \tag{11}$$

$$\text{S-argmax}(\mathbf{x})_k = \frac{\exp(x_k - m)}{\sum_{j=1}^{N}\exp(x_j - m)} \tag{12}$$

where $m = \max(\mathbf{x})$. Intuitively, S-$\max(\mathbf{x})$ returns a value close to m, while remaining differentiable in all components of $\mathbf{x}$. The S-argmax corresponds to the standard softmax function, producing a probability distribution over indices that reflects their relative proximity to the maximum.

Differences with respect to [7]. Our Differentiable Viterbi Layer (DVL) builds on the general framework of differentiable dynamic programming proposed by Mensch and Blondel [7], but differs in several important respects. First, while [7] introduced a unified approach to smoothing dynamic programs and enabled end-to-end training of both transition and emission potentials, our DVL does not introduce new trainable parameters: transition probabilities are fixed by the procedural knowledge graph (PKG), and emission probabilities are provided by upstream modules. Second, we explicitly introduce *soft backpointer distributions* and their recursive composition into a *soft plan*, which serves as a differentiable analogue of the discrete Viterbi backtrace. In summary, our contribution is a re-designed decoding-only layer that leverages fixed structural knowledge to produce differentiable plans, enabling gradient flow through decoding without learning dynamic programming parameters.

1.D. Visual Encoding (fn 5)

Following prior work, we employ S3D [13] as our visual backbone to extract spatiotemporal features from start and goal video states. The resulting representations are then passed through a learned projection layer, which adapts the backbone features to the dimensionality required by our planning model.

1.E. Training (fn 6)

ViterbiPlanNet is trained end-to-end by minimizing a composite loss function $\mathcal{L}$ defined as a sum of three distinct loss components with equal weights.

Visual-Semantic Alignment Loss ($\mathcal{L}_{\text{align}}$). To encourage the model to learn a structured state space, we align visual representations with textual descriptions of their corresponding procedural states as in [9]. Following the idea of SCHEMA [9], for each action in the vocabulary we obtain a set of natural language descriptions of its *before-state* and *after-state* using the same prompt and model used in SCHEMA [9]. We denote this set of descriptions as $\mathcal{A}$. These descriptions capture discriminative object attributes (e.g., "the pan has no onion on it" before *add onion*).

Formally, given the encoded start state v_s^{enc}, the goal state v_g^{enc}, and a textual description $d \in \mathcal{A}$ encoded by a frozen language model into an embedding d^{enc}, we compute cosine similarities between the visual embeddings and all candidate textual descriptions:

$$\text{sim}(v_s^{enc}, d^{enc}) = \frac{v_s^{enc}\cdot d^{enc}}{\|v_s^{enc}\|\|d^{enc}\|}, \tag{13}$$

$$\text{sim}(v_g^{enc}, d^{enc}) = \frac{v_g^{enc}\cdot d^{enc}}{\|v_g^{enc}\|\|d^{enc}\|}. \tag{14}$$

During training, the *positive samples* are the textual descriptions corresponding to the ground-truth first action's before-state (for v_s) and the ground-truth last action's after-state (for v_g). All other descriptions act as negatives. We adopt a contrastive cross-entropy loss as in [9], which encourages the visual embeddings to be close to their correct textual descriptions while being far from incorrect ones:

$$\begin{aligned}\mathcal{L}_{align} = \\ &-\log\frac{\exp(sim(v_s^{enc}, d_+^{enc}))}{\sum_{d\in\mathcal{A}}\exp(sim(v_s^{enc}, d^{enc}))} \\ &-\log\frac{\exp(sim(v_g^{enc}, d_+^{enc}))}{\sum_{d\in\mathcal{A}}\exp(sim(v_g^{enc}, d^{enc}))}.\end{aligned} \tag{15}$$

where d_+^{enc} denotes the encoding of the positive samples. This objective explicitly grounds the visual encoder in the semantics of procedural states, ensuring that the learned visual features capture the causal state changes relevant to the procedure.

Task Classification Loss ($\mathcal{L}_{\text{task}}$). Let N_{tasks} denote the total number of possible tasks in the dataset. We represent the ground-truth task label as a one-hot vector $c \in \{0,1\}^{N_{tasks}}$, where $c_n = 1$ if the procedure belongs to class n and 0 otherwise. As shown in the architecture, the auxiliary *Task Head* takes the encoded visual features $(v_s^{\text{enc}}, v_g^{\text{enc}})$ as input and outputs a prediction vector $\hat{c} \in \mathbb{R}^{N_{tasks}}$, where $\hat{c}_n$ is the predicted score for class n (for simplicity $\hat{c}$ is not labeled in the figure and is simply depicted as a square before $\mathcal{L}_{task}$). This auxiliary prediction provides contextual information that implicitly guides the planning process.

To train the Task Head, we minimize the Mean Squared Error (MSE) between the predicted scores and the one-hot ground-truth labels:

$$\mathcal{L}_{task} = \frac{1}{N_{tasks}}\sum_{n=1}^{N_{tasks}}\left(\hat{c}_n - c_n\right)^2. \tag{16}$$

Planning Loss ($\mathcal{L}_{\text{plan}}$). The central objective of our framework is to learn to generate the correct sequence of actions. The final output of the Structured Decoding module,

Table 1. Ablation of Viterbi components on CrossTask for $T \in \{4, 5, 6\}$.

	Train	Inference		Metrics (%) ↑		
	DVL	DVL	VD	SR	mAcc	mIoU
Horizon $T = 4$						
1	×	×	×	18.93 ± 0.58	55.12 ± 0.46	79.87 ± 0.26
2	×	×	✓	18.64 ± 0.75	55.00 ± 0.38	79.78 ± 0.18
3	×	✓	×	21.54 ± 0.50	53.19 ± 0.46	79.90 ± 0.28
4	×	✓	✓	19.92 ± 0.18	52.09 ± 0.43	79.78 ± 0.34
5	✓	×	×	6.13 ± 0.31	44.71 ± 0.13	69.70 ± 0.56
6	✓	×	✓	23.46 ± 0.20	**57.13** ± 0.35	81.05 ± 0.38
7	✓	✓	×	24.30 ± 0.66	56.42 ± 0.10	80.93 ± 0.48
8	✓	✓	✓	**24.64** ± 0.30	57.00 ± 0.42	**81.18** ± 0.44
Improvement w.r.t. conf. 1				5.71 ± 0.64	1.88 ± 0.61	1.31 ± 0.52
Horizon $T = 5$						
1	×	×	×	10.21 ± 0.08	50.49 ± 0.64	77.49 ± 0.44
2	×	×	✓	9.89 ± 0.08	50.44 ± 0.59	77.42 ± 0.44
3	×	✓	×	13.32 ± 0.26	48.64 ± 0.51	77.70 ± 0.37
4	×	✓	✓	12.27 ± 0.19	47.78 ± 0.63	77.51 ± 0.28
5	✓	×	×	1.77 ± 0.31	38.57 ± 0.58	65.48 ± 1.08
6	✓	×	✓	14.86 ± 0.36	**53.63** ± 0.16	**79.58** ± 0.23
7	✓	✓	×	15.69 ± 0.55	52.07 ± 0.40	79.18 ± 0.33
8	✓	✓	✓	**15.97** ± 0.17	53.30 ± 0.29	79.56 ± 0.27
Improvement w.r.t. conf. 1				5.76 ± 0.18	2.81 ± 0.71	2.07 ± 0.54
Horizon $T = 6$						
1	×	×	×	4.70 ± 0.40	45.73 ± 0.91	76.15 ± 0.33
2	×	×	✓	4.48 ± 0.30	45.61 ± 1.10	76.02 ± 0.33
3	×	✓	×	7.59 ± 0.32	43.66 ± 0.79	76.19 ± 0.24
4	×	✓	✓	7.20 ± 0.26	43.12 ± 0.67	76.17 ± 0.22
5	✓	×	×	0.50 ± 0.19	33.12 ± 0.75	61.08 ± 1.96
6	✓	×	✓	9.18 ± 0.26	**49.71** ± 0.36	**78.17** ± 0.17
7	✓	✓	×	9.99 ± 0.22	48.06 ± 0.37	77.58 ± 0.25
8	✓	✓	✓	**10.37** ± 0.22	49.25 ± 0.54	78.01 ± 0.21
Improvement w.r.t. conf. 1				5.67 ± 0.43	3.52 ± 1.00	1.86 ± 0.40

$\tilde{\pi} \in \mathbb{R}^{T \times N}$, represents the refined score distribution over all N possible actions at each of the T time steps. We supervise these predictions against the one-hot encoded ground-truth plan $\tilde{\pi}^{GT} \in \{0, 1\}^{T \times N}$, where $\tilde{\pi}^{GT}[t, n] = 1$ if the ground-truth action at time step t is $K_n \in \mathcal{K}$ and 0 otherwise. The Mean Squared Error (MSE) loss is then defined as:

$$\mathcal{L}_{\text{plan}} = \frac{1}{T} \sum_{t=1}^{T} \left(\tilde{\pi}_t - \tilde{\pi}_t^{GT} \right)^2 . \tag{17}$$

Minimizing $\mathcal{L}_{\text{plan}}$ encourages the model to produce action distributions that closely match the target one-hot plan. With this the model learns accurate procedural sequences while being constrained by the structural priors encoded in the Procedural Knowledge Graph, which are enforced through the Differentiable Viterbi layers.

Overall Objective. The final training loss is a sum of the three components with equal weights:

$$\mathcal{L} = \mathcal{L}_{plan} + \mathcal{L}_{align} + \mathcal{L}_{task}. \tag{18}$$

2. Experiments and Results

2.A. Discussion on mIoU metric

As discussed in MTID [16], the definition of mIoU varies across the literature. The conventional *set-based* formulation treats each sequence as an unordered set of unique actions:

$$\text{mIoU}_{\text{set}} = \frac{100}{N} \sum_{i=1}^{N} \frac{|\tilde{\pi}_i \cap \tilde{\pi}_i^{GT}|}{|\tilde{\pi}_i \cup \tilde{\pi}_i^{GT}|}, \tag{19}$$

where $\tilde{\pi}_i$ and $\tilde{\pi}_i^{GT}$ denote the predicted and ground-truth actions for sequence i, and N is the number of sequences. While this formulation captures the overall action coverage, it ignores temporal order and repeated actions. This limitation can lead to inflated scores in procedural settings where sequence structure and frequency of actions are crucial (e.g., the action "*stir mixture*" may occur multiple times).

To address this limitation, we adopt the *element-wise* (mask-based) IoU formulation introduced in SCHEMA [9]. In this variant, the IoU is computed independently for each sequence by comparing the predicted and ground-truth binary masks along the temporal dimension:

$$\text{mIoU}_{\text{mask}} = \frac{100}{N} \sum_{i=1}^{N} \frac{\sum_{t=1}^{T} \left[\tilde{\pi}_{i,t} \wedge \tilde{\pi}_{i,t}^{GT} \right]}{\sum_{t=1}^{T} \left[\tilde{\pi}_{i,t} \vee \tilde{\pi}_{i,t}^{GT} \right] + \varepsilon}, \tag{20}$$

where $\tilde{\pi}_{i,t}$ and $\tilde{\pi}_{i,t}^{GT}$ denote the predicted and ground-truth action for sequence i at time step t, ε is a small constant for numerical stability, and $[\cdot]$ is 1 if the logical operation between $\tilde{\pi}_{i,t}$ and $\tilde{\pi}_{i,t}^{GT}$ is true.

Unlike the set-based IoU, this element-wise metric preserves both temporal order and action frequency, offering a more faithful evaluation of sequence prediction, especially in tasks where ordering and repetition are essential.

2.B. Bootstrap Procedure for Statistical Significance (fn 6)

Bootstrap Confidence Intervals for Single-Model Estimates. To report uncertainty for each model, we compute a bootstrap confidence interval over the scores obtained from the multiple training seeds. For a given metric (SR, mAcc, or mIoU) and a fixed planning horizon T, let $\{x_1, \ldots, x_n\}$ denote the scores obtained from n different seeds. The empirical mean is defined as:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i. \tag{21}$$

To estimate the uncertainty around $\bar{x}$, we perform K bootstrap resamplings (with $K = 10^2$ in all experiments). Each bootstrap replicate is constructed by sampling with replacement from the original set:

$$X^* = \{x_1^*, \ldots, x_n^*\}, \qquad x_i^* \sim \{x_1, \ldots, x_n\}. \tag{22}$$

For each bootstrap sample, we compute its mean:

$$\mu_k^* \;=\; \frac{1}{n}\sum_{i=1}^{n} x_i^*, \qquad k = 1, \ldots, K. \tag{23}$$

The distribution of the bootstrap means $\{\mu_k^*\}_{k=1}^K$ is then used to estimate a 90% confidence interval by taking the 5th and 95th percentiles:

$$\mathrm{CI}_{90} \;=\; \big[\mu^*_{(5\%)},\ \mu^*_{(95\%)}\big]. \tag{24}$$

In the tables, each metric is reported in the form $\bar{x} \pm \mathrm{CI}$, where $\mathrm{CI} = \mu^*_{(95\%)} - \mu^*_{(5\%)}$ is the confidence interval width.

Comparison between two models. To quantify whether the performance differences between two models are statistically significant, we employ a non-parametric bootstrap procedure over the five different training seeds used for each experiment. Let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$ denote the seed-level scores (e.g., SR, mAcc, mIoU) for two models, with $n = 5$. The observed difference in means is defined as:

$$\Delta_{\mathrm{obs}} \;=\; \frac{1}{n}\sum_{i=1}^{n} a_i \;-\; \frac{1}{n}\sum_{i=1}^{n} b_i. \tag{25}$$

To assess its reliability, we generate $K = 10^3$ bootstrap replicates. Each replicate samples (with replacement) the sets:

$$A^* = \{a_1^*, \ldots, a_n^*\}, \qquad B^* = \{b_1^*, \ldots, b_n^*\}, \tag{26}$$

where $a_i^* \sim A$ and $b_i^* \sim B$. For each pair of resampled sets, we compute the bootstrap difference

$$\Delta_k^* \;=\; \frac{1}{n}\sum_{i=1}^{n} a_i^* \;-\; \frac{1}{n}\sum_{i=1}^{n} b_i^*, \quad k = 1, \ldots, K. \tag{27}$$

The empirical distribution of $\{\Delta_k^*\}_{k=1}^K$ is used to obtain a 90% confidence interval:

$$\mathrm{CI}_{90} \;=\; \big[\,\Delta^*_{(5\%)}\,,\ \Delta^*_{(95\%)}\,\big], \tag{28}$$

where $\Delta^*_{(p\%)}$ denotes the p-th percentile of the bootstrap distribution. Improvement in each table is reported in the form $\Delta_{\mathrm{obs}} \pm \mathrm{CI}$. For Δ_{obs} each a_i is sampled from our model and each b_i is sampled from the second best model (or from the specified model/configuration). Here, $\mathrm{CI} = \Delta^*_{(95\%)} - \Delta^*_{(5\%)}$. We deem each improvement *statistically significant* if and only if:

$$0 \notin \mathrm{CI}_{90}, \tag{29}$$

i.e., both endpoints have the same sign.

2.C. Ablations on CrossTask (fn 8)

Importance of Structure-Aware Training. Table 1 reports results for $T \in \{4, 5, 6\}$ and reveals the same trends observed for T=3. Across all horizons, three consistent patterns emerge. Together, these findings generalize our conclusions from T=3 and demonstrate that the benefits of training using DVL persist across longer planning horizons.

Structure-aware training is effective. Models trained with the Differentiable Viterbi Layer (DVL) (configurations 6-8) outperform their counterparts trained without DVL (configurations 1-4). The absolute gains in SR remain remarkably stable across horizons ($\approx$ 5.7%), indicating that the benefits of structure-aware learning scale reliably with sequence length. In contrast, enabling VD or DVL *only at inference* (configurations 2-4) yields negligible improvements over the baseline, confirming that most of the gains originate from structured training rather than test-time post-processing.

DVL learns meaningful emissions. For every horizon, decoding the learned emissions with a row-wise argmax (configuration 5) leads to a substantial drop in performance, mirroring the behaviour observed for T=3. This confirms that emissions learned with DVL represent distributions over latent states rather than direct action scores, and therefore require a structured decoding procedure. When these emissions are decoded through VD or DVL (configurations 6-8), performance recovers and consistently exceeds the non-DVL baseline.

DVL is Backward-Compatible with standard VD. Replacing DVL with standard VD at inference (configuration 6 vs. 7) results in comparable performance across all metrics and horizons, showing that VD can operate effectively on emissions produced by DVL-trained models. Adding VD on top of DVL (configuration 8) produces only marginal differences. These results confirm that the main advantage stems from structure-aware *training*, with VD and DVL playing largely interchangeable roles at inference.

Memorization and Sample Efficiency. Figures 2, 3, 4 show performance as a function of training data on CrossTask for $T \in \{4, 5, 6\}$. They confirm the same patterns observed for T=3 in the main paper: ViterbiPlanNet is consistently more sample-efficient than SCHEMA. Across all horizons, ViterbiPlanNet achieves higher success rates when trained with limited data (e.g., $5\% - 25\%$ of the training set), while SCHEMA requires substantially more examples to reach comparable performance. This gap progressively narrows as the training set grows, indicating that SCHEMA increasingly benefits from memorization as more trajectories are available.

When the PKG is removed (dashed lines) the results again follow the same trend across all horizons: SCHEMA outperforms the Base Model (corresponding to configuration 1 in Table 1) due to its more flexible transformer-based

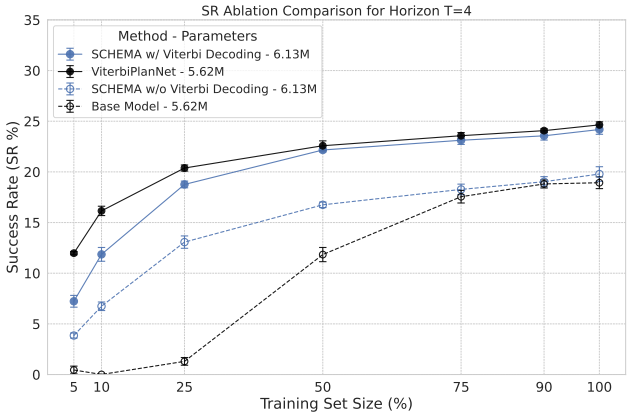

Figure 2. Performance as a function of training data on CrossTask for $T = 4$.

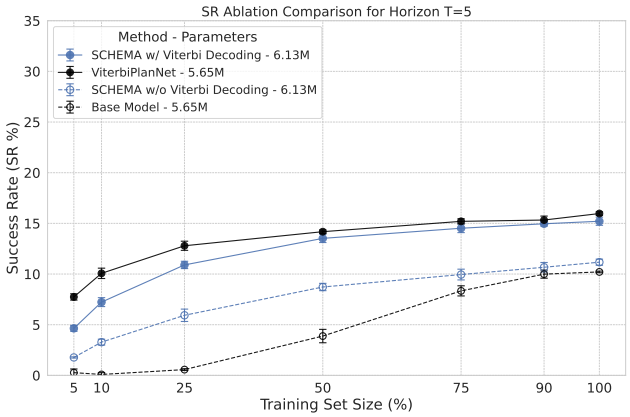

Figure 3. Performance as a function of training data on CrossTask for $T = 5$.

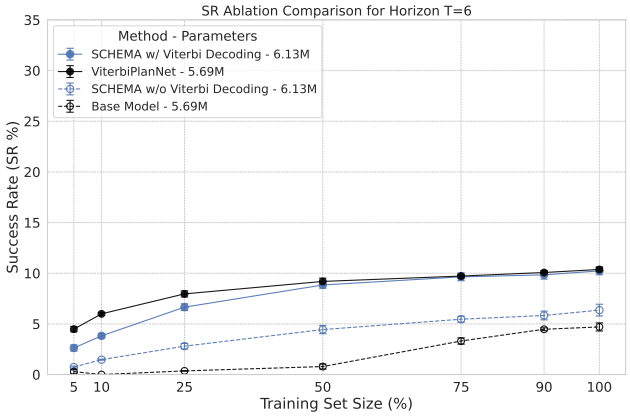

Figure 4. Performance as a function of training data on CrossTask for $T = 6$.

Table 2. SR ↑ (%) with and without PKG on CrossTask for Horizons $T \in \{4, 5, 6\}$.

Method	PKG Use	w/o PKG	w/ PKG	Improv.
Horizon $T = 4$				
KEPP [8]	Conditioning	22.57 ± 0.52	22.34 ± 0.43	-0.23 ± 0.69
PlanLLM [15]	Post-processing	19.90 ± 0.40	22.91 ± 1.39	3.01 ± 1.42
SCHEMA [9]	Post-processing	19.79 ± 0.72	24.18 ± 0.47	4.39 ± 0.91
ViterbiPlanNet	Guided Training	18.93 ± 0.58	24.64 ± 0.30	5.71 ± 0.64
Horizon $T = 5$				
KEPP [8]	Conditioning	13.39 ± 0.32	13.36 ± 1.06	-0.03 ± 1.08
PlanLLM [15]	Post-processing	12.06 ± 0.40	14.89 ± 0.24	2.83 ± 0.45
SCHEMA [9]	Post-processing	11.17 ± 0.30	15.21 ± 0.40	4.04 ± 0.51
ViterbiPlanNet	Guided Training	10.21 ± 0.08	15.97 ± 0.18	5.76 ± 0.19
Horizon $T = 6$				
KEPP [8]	Conditioning	7.91 ± 0.66	8.21 ± 0.22	0.30 ± 0.70
PlanLLM [15]	Post-processing	7.03 ± 0.38	8.98 ± 0.97	1.95 ± 1.04
SCHEMA [9]	Post-processing	6.36 ± 0.58	10.23 ± 0.38	3.87 ± 0.67
ViterbiPlanNet	Guided Training	4.70 ± 0.40	10.37 ± 0.22	5.67 ± 0.43

architecture, which facilitates memorization of procedural patterns. Importantly, the Base Model and ViterbiPlanNet share the same architecture and parameter count, so the consistent improvement of ViterbiPlanNet over the Base Model is entirely attributable to its PKG-aware structured training, rather than additional capacity. These observations, stable across T=4 and T=5 as well as T=6, demonstrate that using the PKG within our differentiable planning module reduces the need for memorization and leads to better sample efficiency across all planning horizons.

Guided Training vs Conditioning and Post-processing. Table 2 extends our comparison of how different methods leverage the PKG to longer horizons ($T \in \{4, 5, 6\}$), revealing the same pattern observed at T=3. Across all horizons, we again find that *all* approaches benefit from the PKG, but *not equally*. Methods that use the PKG only as a conditioning signal (KEPP) or as a post-processing constraint (PlanLLM and SCHEMA) exhibit modest gains: KEPP provides almost no improvement, while PlanLLM and SCHEMA obtain consistent but moderate increases. In contrast, ViterbiPlanNet, which incorporates the PKG directly into the training objective through guided training with a Differentiable Viterbi Layer, achieves the *largest* improvement at every horizon. Specifically, ViterbiPlanNet improves by $+5.71\%$, $+5.76\%$, and $+5.67\%$ SR for T=4, 5, 6 respectively, substantially surpassing all alternatives.

These results confirm that learning through the PKG is consistently more effective than conditioning on it or applying it only as a test-time constraint. In particular, the advantage of ViterbiPlanNet persists as the planning horizon grows, indicating that guided training extracts a stronger procedural signal and scales more robustly to more challenging horizons.

Effect of Task Supervision. We assess the role of task supervision by removing the task head and loss during training for all methods. Table 3 reports results across multiple planning horizons ($T \in \{3, 4, 5, 6\}$). A clear pat-

Table 3. SR ↑ (%) with and without Task Supervision (Task S.) for Horizons $T \in \{3, 4, 5, 6\}$.

Method	w/ Task S.	w/o Task S.
Horizon $T = 3$		
PDPP [12]	36.73 ± 0.59	8.39 ± 8.02
PlanLLM [15]	36.84 ± 1.21	15.37 ± 14.09
SCHEMA [9]	37.24 ± 0.60	37.00 ± 0.27
ViterbiPlanNet	38.45 ± 0.32	38.32 ± 0.12
Horizon $T = 4$		
PDPP [12]	21.47 ± 2.09	5.24 ± 1.23
PlanLLM [15]	23.25 ± 0.38	4.97 ± 3.10
SCHEMA [9]	24.18 ± 0.47	23.24 ± 0.98
ViterbiPlanNet	24.64 ± 0.30	24.68 ± 0.26
Horizon $T = 5$		
PDPP [12]	13.79 ± 0.21	1.51 ± 1.33
PlanLLM [15]	14.88 ± 0.28	5.33 ± 7.43
SCHEMA [9]	15.21 ± 0.40	14.80 ± 0.32
ViterbiPlanNet	15.97 ± 0.17	15.81 ± 0.21
Horizon $T = 6$		
PDPP	8.68 ± 0.63	0.46 ± 0.22
PlanLLM	9.23 ± 0.17	2.82 ± 3.35
SCHEMA	10.23 ± 0.38	9.32 ± 0.46
ViterbiPlanNet	10.37 ± 0.22	10.18 ± 0.29

tern emerges. Methods whose generation is tightly coupled to the task identity such as PDPP and PlanLLM suffer dramatic performance drops when task labels are removed (e.g., SR at T=3 for PDPP $36.73\% \rightarrow 8.39\%$ and PlanLLM $36.84\% \rightarrow 15.37\%$). This happens often with large variance, indicating strong dependence on explicit task conditioning. In contrast, SCHEMA remains remarkably stable across all horizons with changes typically below 1%, reflecting the fact that its LLM-derived procedural memory implicitly encodes task-specific structure even without task supervision.

ViterbiPlanNet exhibits the same robustness: performance remains unchanged when task supervision is removed (e.g., $38.45\% \rightarrow 38.32\%$ at T=3, with similar behaviour for larger horizons). This stability stems from the differentiable Viterbi Layer (DVL), which internalizes procedural constraints directly from the PKG and enforces them throughout training. As a result, ViterbiPlanNet learns task-aware structural priors without requiring explicit task annotations.

2.D. Comparisons with the State of the Art (fn 8)

Performance on Different Planning Horizons. Table 4 reports the full comparison across all datasets for horizons beyond those shown in the main paper. The trends observed for T=3 and T=4 remain fully consistent at larger horizons.

ViterbiPlanNet achieves the highest Success Rate (SR) across all settings. Although performance gaps naturally shrink as the task becomes harder and all methods degrade, ViterbiPlanNet continues to match or surpass the strongest baselines, typically SCHEMA or PlanLLM, and remains the top performer in SR, demonstrating robust long-horizon sequential modeling.

Step-level metrics (mAcc and mIoU) remain comparable to other approaches. Similar to shorter horizons, SCHEMA and PlanLLM occasionally report slightly higher mAcc or mIoU on COIN, but differences are small and not statistically significant. This confirms that ViterbiPlanNet's emphasis on global procedural consistency does not compromise local accuracy, even for longer sequences.

In-context LLM/VLM models achieve limited performance. At T=5 and especially T=6, Qwen2.5-VL-32B, Qwen LLMs, and Gemini 2.5 Pro exhibit large drops in SR, often falling below the simple PKG beam-search baseline. This highlights that zero-shot prompting strategies struggle to maintain coherent multi-step reasoning as planning depth increases.

Cross-Horizon Consistency. Tables 6 and 7 report cross-horizon consistency results on COIN and NIV, extending the analysis from the main paper. The same trends observed on CrossTask clearly emerge across both datasets. On COIN, LLMs and VLMs exhibit limited robustness when evaluated at shorter horizons after training at T=6, with performance often collapsing as the required subsequence length decreases. Learning-based approaches such as PDPP, KEPP, and PlanLLM show slightly better stability but still suffer from noticeable degradation, particularly when moving from 6→3. SCHEMA stands out as the strongest baseline, yet ViterbiPlanNet consistently surpasses it by substantial margins across all horizon reductions, achieving gains of up to $+4.44\%$ SR.

On NIV, the role of explicit procedural structure is particularly evident. PKG beam search stands out as the strongest non-learning baseline and, in fact, the second-best overall method, clearly demonstrating the high importance of the PKG signal on this dataset. This indicates that NIV strongly benefits from structured graph-based priors. Importantly, ViterbiPlanNet is the only model that fully leverages this signal. The improvement is statistically significant for the 6→3 setting ($+3.85\%$ SR), while for 6→4 and 6→5 the gains remain positive but not statistically conclusive. These results underscore that ViterbiPlanNet is uniquely capable of leveraging the PKG to achieve robust horizon-invariant planning behavior.

Table 4. Comparison with the state of the art. **Best** and second-best results are highlighted for each metric within each time horizon. Statistically significant performance differences (i.e., cases in which the confidence interval does not include zero) are marked in yellow.

T	Method	CrossTask				COIN				NIV			
		SR ↑ (%)	mAcc ↑ (%)	mIoU ↑ (%)	Params (M)	SR ↑ (%)	mAcc ↑ (%)	mIoU ↑ (%)	Params (M)	SR ↑ (%)	mAcc ↑ (%)	mIoU ↑ (%)	Params (M)
3	Qwen2.5-VL-32B [1]	11.48	36.35	69.52	32,000	3.65	17.51	52.10	32,000	7.41	27.65	59.73	32,000
	Qwen2.5-32B [10]	25.14	56.10	80.92	32,000	14.97	36.34	78.74	32,000	24.07	43.46	71.88	32,000
	Gemini 2.5 Pro [3]	29.18	57.90	81.48	>100,000	17.02	38.87	78.73	>100,000	24.07	43.46	71.86	>100,000
	Qwen3-30B [14]	23.37	55.96	81.16	30,000	14.52	36.56	78.07	30,000	24.81	42.84	70.80	30,000
	Qwen3-30B [14] + PKG	23.31	56.15	81.06	30,000	14.63	36.53	78.11	30,000	25.19	43.95	71.98	30,000
	PKG beam search	22.38 ± 0.26	55.74 ± 0.25	80.92 ± 0.26	41.87	13.32 ± 0.34	37.42 ± 1.19	78.93 ± 2.06	42.90	24.96 ± 1.93	43.46 ± 2.42	72.18 ± 0.55	41.74
	PDPP [12]	36.73 ± 0.59	61.96 ± 0.59	83.20 ± 0.33	41.87	22.37 ± 0.57	44.60 ± 0.16	83.00 ± 0.42	42.90	26.52 ± 1.56	45.58 ± 1.85	**74.89** ± 0.85	41.74
	KEPP [8]	34.93 ± 2.60	60.34 ± 1.61	82.67 ± 0.69	42.18	13.85 ± 7.49	28.40 ± 12.26	62.54 ± 14.35	44.66	27.56 ± 1.48	45.93 ± 2.37	74.36 ± 0.97	41.86
	PlanLLM [15]	36.84 ± 1.21	61.56 ± 1.03	83.23 ± 0.53	384.94	33.44 ± 0.15	**51.05** ± 0.46	**84.66** ± 0.41	386.43	30.00 ± 1.41	44.35 ± 2.52	73.60 ± 1.66	384.77
	SCHEMA [9]	37.24 ± 0.60	62.69 ± 0.28	**83.94** ± 0.23	6.13	32.89 ± 0.61	50.84 ± 0.47	83.98 ± 0.67	6.28	26.30 ± 1.49	42.77 ± 2.12	73.04 ± 1.42	6.12
	ViterbiPlanNet	**38.45** ± 0.32	**63.07** ± 0.17	83.89 ± 0.16	5.57	**33.99** ± 0.23	50.87 ± 0.17	83.88 ± 0.31	6.67	**32.37** ± 0.96	**46.96** ± 1.75	73.85 ± 0.85	5.48
	Improvement	+1.21 ± 0.69	+0.38 ± 0.34	-0.05 ± 0.27		+0.55 ± 0.27	-0.18 ± 0.49	-0.78 ± 0.50		+2.37 ± 1.63	+1.04 ± 3.06	-1.04 ± 1.22	
4	Qwen2.5-VL-32B [1]	5.56	31.22	66.31	32,000	1.87	17.05	55.66	32,000	5.26	28.84	60.21	32,000
	Qwen2.5-32B [10]	9.22	46.32	76.15	32,000	4.98	27.45	71.64	32,000	23.25	41.89	73.91	32,000
	Gemini 2.5 Pro [3]	14.00	51.33	78.58	>100,000	8.10	31.90	71.70	>100,000	22.37	40.35	73.05	>100,000
	Qwen3-30B [14]	10.59	49.06	78.03	30,000	4.64	28.85	70.45	30,000	22.37	41.23	73.90	30,000
	Qwen3-30B [14] + PKG	10.96	48.77	77.48	30,000	4.78	29.00	71.04	30,000	21.93	41.67	74.43	30,000
	PKG beam search	9.30 ± 0.22	47.65 ± 0.54	78.25 ± 0.42	41.87	5.14 ± 0.60	31.29 ± 3.64	74.26 ± 5.38	42.90	21.23 ± 0.96	40.86 ± 0.83	72.69 ± 0.75	41.74
	PDPP [12]	21.47 ± 2.09	55.66 ± 1.64	80.68 ± 0.83	41.87	15.21 ± 0.34	41.01 ± 0.32	81.64 ± 0.48	42.90	21.40 ± 0.53	40.20 ± 2.00	72.82 ± 1.84	41.74
	KEPP [8]	22.34 ± 0.43	55.24 ± 0.30	80.58 ± 0.25	42.18	15.20 ± 1.27	33.39 ± 0.73	67.79 ± 1.29	44.66	22.54 ± 1.93	42.46 ± 1.49	73.11 ± 0.94	41.86
	PlanLLM [15]	22.91 ± 1.39	55.29 ± 1.54	81.03 ± 0.47	384.94	23.19 ± 0.32	**45.70** ± 0.33	**83.44** ± 0.39	386.43	23.42 ± 1.40	41.95 ± 2.81	72.32 ± 0.91	384.77
	SCHEMA [9]	24.18 ± 0.47	**57.02** ± 0.64	**81.46** ± 0.19	6.13	22.33 ± 0.92	45.21 ± 1.05	82.93 ± 0.25	6.28	24.39 ± 1.84	41.14 ± 3.62	73.13 ± 1.97	6.12
	ViterbiPlanNet	**24.64** ± 0.30	57.00 ± 0.42	81.18 ± 0.44	5.60	**23.92** ± 0.29	45.63 ± 0.55	82.56 ± 0.44	6.87	**27.54** ± 0.70	**45.55** ± 1.89	**74.71** ± 1.19	5.50
	Improvement	+0.46 ± 0.61	-0.02 ± 0.78	-0.29 ± 0.49		+0.73 ± 0.44	-0.08 ± 0.62	-0.88 ± 0.59		+3.15 ± 1.93	+3.09 ± 2.43	+1.58 ± 2.37	
5	Qwen2.5-VL-32B [1]	2.27	24.53	63.65	32,000	1.10	16.75	57.21	32,000	1.07	29.30	61.98	32,000
	Qwen2.5-32B [10]	3.77	37.58	72.09	32,000	4.04	28.48	74.77	32,000	18.72	43.32	73.47	32,000
	Gemini 2.5 Pro [3]	4.35	42.76	76.47	>100,000	7.89	32.48	75.22	>100,000	18.72	40.86	71.71	>100,000
	Qwen3-30B [14]	4.09	39.14	72.83	30,000	3.65	28.46	72.44	30,000	19.25	43.21	72.90	30,000
	Qwen3-30B [14] + PKG	4.77	39.83	73.17	30,000	3.55	29.08	74.06	30,000	20.32	45.35	74.25	30,000
	PKG beam search	5.35 ± 0.03	42.98 ± 0.69	76.75 ± 0.54	41.87	2.91 ± 0.19	28.93 ± 0.93	73.61 ± 1.92	42.90	18.40 ± 1.18	42.65 ± 1.48	74.00 ± 1.58	41.74
	PDPP [12]	13.79 ± 0.21	52.31 ± 0.29	79.21 ± 0.26	41.87	11.42 ± 0.49	37.23 ± 0.34	80.84 ± 0.47	42.90	19.04 ± 2.57	**44.56** ± 2.97	**75.73** ± 1.35	41.74
	KEPP [8]	13.36 ± 1.06	51.27 ± 0.73	78.69 ± 0.62	42.18	12.14 ± 0.35	32.28 ± 0.57	69.19 ± 0.94	44.66	21.07 ± 1.39	44.36 ± 2.25	74.93 ± 1.42	41.86
	PlanLLM [15]	14.89 ± 0.24	51.16 ± 0.70	78.97 ± 0.35	384.94	**16.15** ± 0.41	**40.29** ± 0.86	**82.21** ± 1.67	386.43	21.93 ± 0.43	42.89 ± 1.22	73.84 ± 1.51	384.77
	SCHEMA [9]	15.21 ± 0.40	52.97 ± 0.24	79.44 ± 0.29	6.13	15.30 ± 0.73	39.47 ± 0.89	81.27 ± 1.09	6.28	19.14 ± 0.97	39.25 ± 3.34	72.97 ± 2.34	6.12
	ViterbiPlanNet	**15.97** ± 0.17	**53.30** ± 0.29	**79.56** ± 0.27	5.64	15.87 ± 0.53	39.42 ± 0.69	81.19 ± 0.99	7.07	**23.10** ± 0.64	42.97 ± 1.99	74.81 ± 1.28	5.51
	Improvement	+0.76 ± 0.43	+0.33 ± 0.40	+0.12 ± 0.40		-0.21 ± 0.70	-0.93 ± 1.29	-1.23 ± 2.04		+1.18 ± 0.86	-1.59 ± 3.47	-0.92 ± 1.82	
6	Qwen2.5-VL-32B [1]	1.21	25.17	63.01	32,000	0.57	16.46	57.35	32,000	3.38	32.09	65.33	32,000
	Qwen2.5-32B [10]	4.00	38.71	73.50	32,000	5.12	27.27	69.84	32,000	16.22	42.23	72.29	32,000
	Gemini 2.5 Pro [3]	3.84	38.55	74.54	>100,000	8.79	30.75	70.24	>100,000	16.89	40.20	70.31	>100,000
	Qwen3-30B [14]	3.40	39.09	74.31	30,000	1.91	25.97	68.30	30,000	16.89	41.89	71.28	30,000
	Qwen3-30B [14] + PKG	3.52	39.52	74.46	30,000	2.67	26.52	68.47	30,000	17.57	43.02	72.20	30,000
	PKG beam search	2.65 ± 0.16	38.64 ± 0.52	74.89 ± 0.54	41.87	0.69 ± 0.18	25.99 ± 0.11	71.25 ± 1.31	42.90	15.41 ± 0.41	42.23 ± 2.68	72.26 ± 1.13	41.74
	PDPP [12]	8.68 ± 0.63	48.50 ± 1.15	78.10 ± 0.59	41.87	9.14 ± 0.44	33.83 ± 0.54	78.42 ± 0.26	42.90	14.19 ± 1.76	43.83 ± 2.43	74.31 ± 1.56	41.74
	KEPP [8]	8.21 ± 0.22	46.45 ± 1.11	76.45 ± 0.74	42.18	10.16 ± 0.53	30.99 ± 0.95	69.40 ± 1.28	44.66	14.05 ± 0.81	41.24 ± 1.49	73.44 ± 1.04	41.86
	PlanLLM [15]	9.04 ± 0.82	45.91 ± 1.24	76.91 ± 0.65	384.94	12.51 ± 0.24	34.97 ± 0.42	78.17 ± 0.48	386.43	16.35 ± 0.81	40.79 ± 1.26	73.52 ± 0.97	384.77
	SCHEMA [9]	10.23 ± 0.38	**49.31** ± 0.49	**78.31** ± 0.34	6.13	**13.16** ± 0.49	**36.41** ± 0.60	79.20 ± 0.85	6.28	15.81 ± 2.97	40.20 ± 3.99	73.46 ± 1.49	6.12
	ViterbiPlanNet	**10.37** ± 0.22	49.25 ± 0.54	78.01 ± 0.21	5.67	13.11 ± 0.35	36.03 ± 0.39	**79.35** ± 0.64	7.27	**18.78** ± 0.81	**45.77** ± 0.83	**75.91** ± 0.52	5.52
	Improvement	+0.14 ± 0.44	-0.06 ± 0.73	-0.30 ± 0.41		-0.05 ± 0.57	-0.38 ± 0.73	+0.15 ± 1.09		+2.43 ± 1.08	+1.94 ± 2.55	+1.60 ± 1.66	

Table 5. Performance comparison of MTID♣ and ViterbiPlanNet♣ across the CrossTask, COIN, and NIV datasets. The **best** and second-best results are highlighted for each metric within each time horizon.

Horizon	Method	CrossTask				COIN				NIV			
		SR ↑ (%)	mAcc ↑ (%)	mIoU ↑ (%)	Params (M)	SR ↑ (%)	mAcc ↑ (%)	mIoU ↑ (%)	Params (M)	SR ↑ (%)	mAcc ↑ (%)	mIoU ↑ (%)	Params (M)
T = 3	MTID♣ [16]	**40.45**	67.19	69.17	1085.20	30.44	**51.70**	59.74	1085.20	28.52	44.44	56.46	1085.20
	ViterbiPlanNet♣	39.75	**67.39**	**76.92**	5.49	**34.42**	51.20	**81.25**	6.01	**34.44**	**48.89**	**95.07**	5.45
T = 4	MTID♣ [16]	**24.76**	60.69	67.67	1085.20	22.74	**49.90**	61.25	1085.20	24.89	44.54	57.46	1085.20
	ViterbiPlanNet♣	24.19	**61.12**	**80.67**	5.52	**24.09**	45.71	**77.84**	6.21	**28.95**	**47.81**	**80.07**	5.46
T = 5	MTID♣ [16]	15.26	-	-	1085.20		-				-		
	ViterbiPlanNet♣	**15.37**	57.10	80.53	5.55		-				-		
T = 6	MTID♣ [16]	**10.30**	-	-	1085.20		-				-		
	ViterbiPlanNet♣	9.68	53.99	86.34	5.57		-				-		

2.E. Comparison with MTID (fn 12)

Table 5 provides the extended comparison between ViterbiPlanNet♣ and MTID♣ [16] across all reported horizons and datasets. Since MTID contains over one billion parameters and is computationally prohibitive to retrain, all MTID results are taken directly from the original paper (if available). To ensure fairness, we adapt ViterbiPlanNet to the MTID evaluation protocol, including the merged CrossTask taxonomy, modified mIoU computation, and PDPP-style feature extraction, and we denote these results with ♣.

Across all datasets and horizons, the trends observed in the main paper remain consistent. Despite being *three or-*

Table 6. Cross-Horizon Consistency results on COIN.

Method	SR ↑ (%) [6 → 3]	SR ↑ (%) [6 → 4]	SR ↑ (%) [6 → 5]
Qwen2.5-VL-32B [1]	3.47	1.75	1.00
Qwen2.5-32B [10]	6.71	6.55	3.75
Gemini 2.5 Pro [3]	5.35	7.20	2.12
Qwen3-30B [14]	5.82	4.31	2.51
Qwen3-30B [14] + PKG	5.80	4.84	2.71
PKG beam search	6.85 ± 0.27	4.39 ± 0.14	1.66 ± 0.20
PDPP [12]	7.66 ± 0.17	6.84 ± 0.36	5.00 ± 0.49
KEPP [8]	5.10 ± 0.80	6.09 ± 1.04	5.85 ± 1.03
PlanLLM [15]	9.48 ± 0.38	8.22 ± 0.27	6.02 ± 0.51
SCHEMA [9]	9.89 ± 0.82	9.30 ± 1.14	7.89 ± 0.64
ViterbiPlanNet	**14.34** ± 0.41	**13.82** ± 0.53	**9.47** ± 0.14
Improvement	+4.45 ± 0.90	+4.52 ± 1.31	+1.58 ± 0.67

Table 7. Cross-Horizon Consistency results on NIV.

Method	SR ↑ (%) [6 → 3]	SR ↑ (%) [6 → 4]	SR ↑ (%) [6 → 5]
Qwen2.5-VL-32B [1]	9.26	5.70	1.60
Qwen2.5-32B [10]	3.33	5.70	8.02
Gemini 2.5 Pro [3]	1.11	6.14	9.63
Qwen3-30B [14]	3.33	5.70	8.02
Qwen3-30B [14] + PKG	4.07	7.46	9.63
PKG beam search	15.19 ± 1.70	14.56 ± 2.46	14.76 ± 2.46
PDPP [12]	13.11 ± 2.22	11.67 ± 1.75	11.23 ± 1.39
KEPP [8]	8.67 ± 1.93	8.77 ± 2.11	11.44 ± 1.93
PlanLLM [15]	9.70 ± 2.59	10.26 ± 1.05	11.55 ± 0.86
SCHEMA [9]	10.37 ± 0.97	11.32 ± 2.19	12.19 ± 2.14
ViterbiPlanNet	**19.04** ± 1.56	**14.74** ± 1.75	**16.90** ± 1.28
Improvement	+3.85 ± 2.37	+0.18 ± 3.33	+2.14 ± 2.67

ders of magnitude smaller (5–7M vs. 1,085M parameters), ViterbiPlanNet achieves performance comparable to MTID in terms of Success Rate and mean Accuracy, and substantially surpasses it in terms of mean IoU. At T=3 and T=4, ViterbiPlanNet delivers large improvements in mIoU, e.g., +7.75 points on CrossTask, +21.51 on COIN, and +38.61 on NIV at T=3, indicating that its predictions are considerably more locally accurate and structurally coherent. For longer horizons (T=5 and T=6), ViterbiPlanNet matches or exceeds the SR reported for MTID.

These results confirm that ViterbiPlanNet achieves competitive or superior performance to MTID while being vastly more efficient, illustrating the benefits of integrating procedural structure directly into the decoding process rather than relying on large diffusion-based models.

3. Additional Studies

3.A. Study on Rigidity of the Markov Assumption (fn 1)

We investigate whether the Markov assumption enforced by Viterbi decoding introduces excessive rigidity during plan generation. Our analysis evaluates three complementary aspects: (i) empirical coverage of the Procedural Knowledge Graph (PKG), (ii) decoding diversity under controlled conditions, and (iii) overlap of failure cases. Quantitative results are summarized in Table 8 and Figure 5.

Table 8. PKG coverage and decoding diversity metrics.

Metric	Dataset/Condition	Result
PKG Coverage (% test sequences with transitions in the graph)	CrossTask (T=3→6)	97.4 → 93.0 (%)
	COIN (T=3→6)	90.8 → 77.0 (%)
	NIV (T=3→6)	86.7 → 72.3 (%)
Decoding Diversity (Entropy H↑, Jensen–Shannon Divergence JSD↓, CrossTask, T=6)	Uniform Emissions	H=0.00, JSD=0.56
	SCHEMA w/o PKG	H=4.39, JSD=0.51
	SCHEMA w/ PKG	H=2.61, JSD=0.39
	ViterbiPlanNet	H=2.13, JSD=0.39

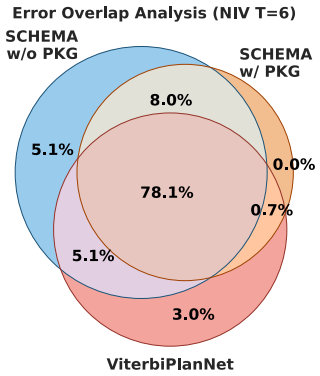

Figure 5. Error overlap analysis on NIV $T = 6$.

PKG Coverage. We first measure PKG coverage, defined as the percentage of test trajectories whose transitions are present in the graph extracted from training data. Coverage naturally decreases with planning horizon and dataset complexity, ranging from 97.4% → 93.0% on CrossTask, 90.8% → 77.0% on COIN, and 86.7% → 72.3% on NIV for T=3→6. Despite longer horizons, coverage remains consistently high ($>$ 72%), indicating that Markovian decoding operates within a broadly permissive transition space. Uncovered transitions correspond primarily to procedures never observed during training, reflecting data sparsity rather than structural limitations of the decoder.

Decoding Diversity. To assess whether Viterbi decoding restricts procedural variability, we evaluate decoding diversity on a controlled subset of CrossTask (T=6), where all sequences share identical start and goal actions but differ in intermediate steps. We report entropy (H↑) and Jensen–Shannon divergence from the ground-truth distribution (JSD↓).

A PKG-only decoder with uniform emissions collapses to a single trajectory (H=0.00), showing that transition constraints alone are insufficient to model procedural variability. Introducing learned emissions substantially increases diversity while maintaining alignment with ground truth. ViterbiPlanNet achieves H=2.13 and JSD=0.39, comparable to SCHEMA with PKG (H=2.61, JSD=0.39). Removing Viterbi decoding (SCHEMA w/o PKG) further increases entropy (H=4.39) but worsens distributional

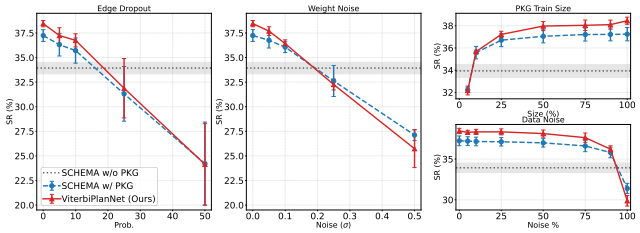

Figure 6. We evaluate performance under progressively corrupted procedural knowledge graphs (PKGs). **Left:** edge dropout obtained by randomly removing graph transitions. **Center:** noisy transition probabilities produced by Gaussian perturbations with variance σ. **Right:** PKGs reconstructed from limited or noisy supervision.

alignment (JSD=0.51), indicating that additional trajectories are largely hallucinated rather than valid alternatives.

These results suggest that the Markov assumption acts as a *structural prior*: it filters implausible transitions while learned emissions retain sufficient flexibility to explore multiple valid plans.

Failure Case Analysis. We further analyze error overlap on NIV at T=6 using the Venn diagram in Figure 5. Adding PKG constraints to SCHEMA corrects 10.2% (5.1+5.1) of failures while introducing only 0.7% new errors, demonstrating that Markovian decoding primarily removes invalid trajectories rather than creating new failure modes. Despite architectural differences, ViterbiPlanNet introduces only 3% new mistakes while correcting 13.1% (8+5.1) and an additional 8% of errors made by SCHEMA without and with PKG, respectively.

Discussion. Overall, the Markov assumption does not impose excessive rigidity. Instead, it provides a permissive yet structured decoding space enabled by high PKG coverage, within which learned emissions steer trajectory selection and recover correct procedural plans. This observation aligns with prior work (e.g., SCHEMA [9], PlanLLM [15]), where Viterbi decoding is routinely adopted as a final inference step.

3.B. Study on Dependence on PKG Quality (fn 2)

We analyze the sensitivity of our method to the quality of the Procedural Knowledge Graph (PKG) by systematically degrading the graph used during training and inference. Specifically, we consider three complementary perturbations: (i) *edge dropout*, obtained by randomly dropping transitions from the graph; (ii) *edge-weight noise*, introduced by adding Gaussian perturbations to transition probabilities; and (iii) *imperfect graph estimation*, where PKGs are reconstructed from limited or noisy supervision (Figure 6).

Across all settings, ViterbiPlanNet exhibits graceful performance degradation. The model remains stable when removing up to 10–15% of graph edges, injecting noise up to $\sigma = 0.15$, or constructing PKGs from as little as 25% of the available training data. Even under extreme label corruption (up to 90% noisy annotations), performance remains competitive and consistently surpasses SCHEMA without PKG constraints (dashed curve).

Although the success rate decreases under severe corruption or extremely limited data, similar degradation trends are observed for SCHEMA with PKG decoding. This indicates that sensitivity primarily stems from inference with incomplete or noisy graphs rather than from our PKG-aware learning formulation.

Table 9. A single model is trained and evaluated using a unified PKG constructed from the union of all datasets. Gray values indicate the average performance of dataset-specific models, while arrows show the performance change when switching to a dataset-independent PKG.

Method [SR ↑ (%)]	$T=3$	$T=4$	$T=5$	$T=6$
SCHEMA [9]	32.1 → 32.7 ±0.4	23.6 → 21.9 ±0.3	16.6 → 14.2 ±0.4	13.1 → 10.2 ±0.2
ViterbiPlanNet	34.9 → **33.3** ±0.4	25.4 → **22.8** ±0.2	18.3 → **15.0** ±0.2	14.1 → **10.8** ±0.2
Improvement	+0.6 ±0.5	+0.8 ±0.4	+0.9 ±0.5	+0.6 ±0.3

3.C. Dataset-Independent PKG (fn 3)

In Table 9 we evaluate whether procedural knowledge can generalize across datasets by training and testing a single model using a unified PKG constructed from the union of all datasets, instead of dataset-specific graphs. Using a shared PKG introduces a modest performance drop compared to the average performance of models trained with dataset-specific graphs ($xx.x \rightarrow zz.z$ overall), reflecting the increased variability and partial mismatch between procedures originating from different domains. Nevertheless, ViterbiPlanNet consistently outperforms SCHEMA across all planning horizons, demonstrating that the learned emissions successfully adapt to heterogeneous procedural structures.

3.D. Intermediate Visual Observations (fn 4)

While planning in instructional videos is typically studied under a weakly supervised setting where only start and end observations are available [8, 9, 12, 15], real-world scenarios often provide additional intermediate visual cues that may refine or revise an ongoing plan. We therefore evaluate whether planning models can effectively incorporate partial observations appearing during execution.

To simulate plan revision, we train and evaluate both ViterbiPlanNet and SCHEMA on CrossTask with planning horizon T=6, providing as input the first three observed actions together with the final observation. This setting supplies partial trajectory evidence while leaving intermediate steps to be inferred, requiring the model to integrate new visual information with procedural priors.

Table 10. Evaluation on the egocentric instructional video dataset EgoPER [6] across planning horizons T=3–6.

Method [SR ↑ (%)]	$T=3$	$T=4$	$T=5$	$T=6$
SCHEMA [9]	29.55 ±1.95	20.78 ±1.94	13.91 ±1.10	12.46 ±1.57
ViterbiPlanNet	**51.84** ±0.95	**48.14** ±0.64	**46.34** ±0.49	**41.98** ±0.92
Improvement	+22.29 ±2.17	+27.36 ±2.04	+32.43 ±1.21	+29.52 ±1.82

Under this protocol, ViterbiPlanNet achieves a substantial improvement in Success Rate, increasing from 10.4 ± 0.2 to 18.3 ± 0.7. In comparison, SCHEMA improves from 10.2 ± 0.4 to 15.7 ± 1.1, resulting in a consistent performance gap in favor of our approach. The larger gain indicates that ViterbiPlanNet uses additional visual context more effectively to refine trajectory predictions.

3.E. Planning in Egocentric Instructional Videos (fn 7)

In Table 10 we further evaluate our approach on **EgoPER** [6], a challenging egocentric instructional video dataset characterized by long-horizon tasks, strong viewpoint changes, and increased visual ambiguity compared to third-person instructional benchmarks. In this setting, action observations are captured from a first-person perspective, making procedural reasoning more difficult due to partial visibility, motion-induced noise, and higher intra-class variability. ViterbiPlanNet achieves substantial improvements over SCHEMA across all planning horizons (T=3–6). Performance gains range from $+22.3\%$ to $+32.4\%$ absolute Success Rate, indicating that structured Markov decoding combined with learned visual emissions remains effective even under severe viewpoint variability.

These results demonstrate that our method generalizes beyond third-person instructional videos and transfers effectively to egocentric planning scenarios. The findings support the hypothesis that procedural structure provides a domain-agnostic inductive bias, enabling robust planning despite substantial visual distribution shifts.

Extending evaluation to robotics and interactive environments with stochastic or branching procedures represents an important direction for future work.

4. Further Experimental Details

4.A. Hyperparameter Configuration

Baseline Configurations. All baselines are implemented following the configurations reported in their original papers, with one exception: PlanLLM [15]. Upon inspection, we found that the released implementation applies a self-attention module over the start, end, and intermediate frames during both training and inference. Since intermediate frames cannot be used at test time, this introduces an unintended information leak. We therefore corrected this behavior by ensuring that PlanLLM attends only to the start and end frames at inference, aligning it with the standard protocol [2] and ensuring a fair comparison.

Table 11. ViterbiPlanNet training configuration.

Component	Name/Value
Visual Backbone	S3D [13]
Optimizer	Adam [5]
Learning Rate	9×10^{-3}
Dropout	0.20
Batch Size	256
Epochs	500
Embedding Dimension (E)	128

ViterbiPlanNet Configuration. Table 11 reports the complete set of hyperparameters used to train ViterbiPlanNet across all datasets. Unless otherwise specified, the same configuration is applied across all planning horizons. Visual representations are extracted using the S3D backbone [13], followed by a projection layer, a lightweight Transformer encoder, an MLP with a Sigmoid activation, and finally the Differentiable Viterbi Layer (DVL). Training is performed with the Adam optimizer [5], using a learning rate of 9×10^{-3}, a dropout rate of 0.20, and a batch size of 256. Models are trained for 500 epochs, which we found sufficient for stable and consistent convergence across datasets and horizons. The embedding dimensionality for action representations is fixed to $E = 128$. The only deviation from this configuration occurs in the experiments reported in Table 6: for COIN at T=6, we employ two concatenated DVL layers during training. This additional depth improves normalization stability and enhances cross-horizon consistency on this particularly challenging dataset.

4.B. LLM and VLM Details (fn 12)

This section details how Large Language Models (LLMs), and Vision-Language Models (VLMs) are employed for planning in instructional videos. Each family of models follows a dedicated strategy, illustrated in Figures 7, 8, and 9.

LLM-Based Sequence Completion. LLMs are used to complete partially observed action sequences given an action taxonomy and several example trajectories. As shown in Figure 7, the model receives: (1) a taxonomy of admissible actions, (2) example sequences demonstrating valid procedural structure, and (3) an incomplete sequence containing the placeholder "-1".

The LLM is instructed to substitute each missing element using only actions from the taxonomy, without generating new actions or explanations. The model may reuse actions multiple times and may adjust the final step if contextually inconsistent.

LLM + PKG Sequence Completion. When a Procedural Knowledge Graph (PKG) is available, we further constrain the LLM with structural priors (Figure 8). In addition to the taxonomy and training examples, the model receives the PKG, which encodes known action dependencies and admissible transitions following [4].

VLM Sequence Completion. For Vision–Language Models, we adopt a two-stage approach (Figure 9).

In the first stage, the VLM is provided with: (1) the action taxonomy, and (2) a video clip depicting the execution of the *start* and *end* actions. The model must identify and return exactly two actions from the taxonomy.

In the second stage, the predicted start and end actions are inserted into Prompt 1, and the VLM is then queried again, now conditioned on the video and the updated prompt, to generate the full action sequence. This two-step strategy enables the VLM to produce a complete plan while remaining grounded in the visual evidence and constrained by the predefined action taxonomy.

Prompts. Prompts 1, 2, and 3 reproduce the exact prompts used for LLMs, LLMs+PKG, and VLMs, respectively, ensuring reproducibility of our experimental setup.

References

[1] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025. 8, 9

[2] 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. 11

[3] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025. 8, 9

[4] Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a graph: Encoding graphs for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. 12

[5] Diederik P. Kingma and Jimmy Lei Ba. A method for stochastic optimization. In *International Conference for Learning Representations*, 2015. 11

[6] Shih-Po Lee, Zijia Lu, Zekun Zhang, Minh Hoai, and Ehsan Elhamifar. Error detection in egocentric procedural task videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18655–18666, 2024. 11

[7] Arthur Mensch and Mathieu Blondel. Differentiable dynamic programming for structured prediction and attention. In *International Conference on Machine Learning*, pages 3462–3471. PMLR, 2018. 2, 3

[8] Kumaranage Ravindu Yasas Nagasinghe, Honglu Zhou, Malitha Gunawardhana, Martin Renqiang Min, Daniel Harari, and Muhammad Haris Khan. Why not use your textbook? knowledge-enhanced procedure planning of instructional videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18816–18826, 2024. 6, 8, 9, 10

[9] Yulei Niu, Wenliang Guo, Long Chen, Xudong Lin, and Shih-Fu Chang. SCHEMA: State CHanges MAtter for procedure planning in instructional videos. In *The Twelfth International Conference on Learning Representations*, 2024. 3, 4, 6, 7, 8, 9, 10, 11

[10] A Yang Qwen, Baosong Yang, B Zhang, B Hui, B Zheng, B Yu, Chengpeng Li, D Liu, F Huang, H Wei, et al. Qwen2. 5 technical report. *arXiv preprint*, 2024. 8, 9

[11] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967. 2

[12] Hanlin Wang, Yilu Wu, Sheng Guo, and Limin Wang. Pdpp: Projected diffusion for procedure planning in instructional videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14836–14845, 2023. 7, 8, 9, 10

[13] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In *Proceedings of the European conference on computer vision (ECCV)*, pages 305–321, 2018. 3, 11

[14] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. 8, 9

[15] Dejie Yang, Zijing Zhao, and Yang Liu. Planllm: Video procedure planning with refinable large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 9166–9174, 2025. 6, 7, 8, 9, 10, 11

[16] Yufan Zhou, Zhaobo Qi, Lingshuai Lin, Junqi Jing, Tingting Chai, Beichen Zhang, Shuhui Wang, and Weigang Zhang. Masked temporal interpolation diffusion for procedure planning in instructional videos. In *The Thirteenth International Conference on Learning Representations*, 2025. 4, 8

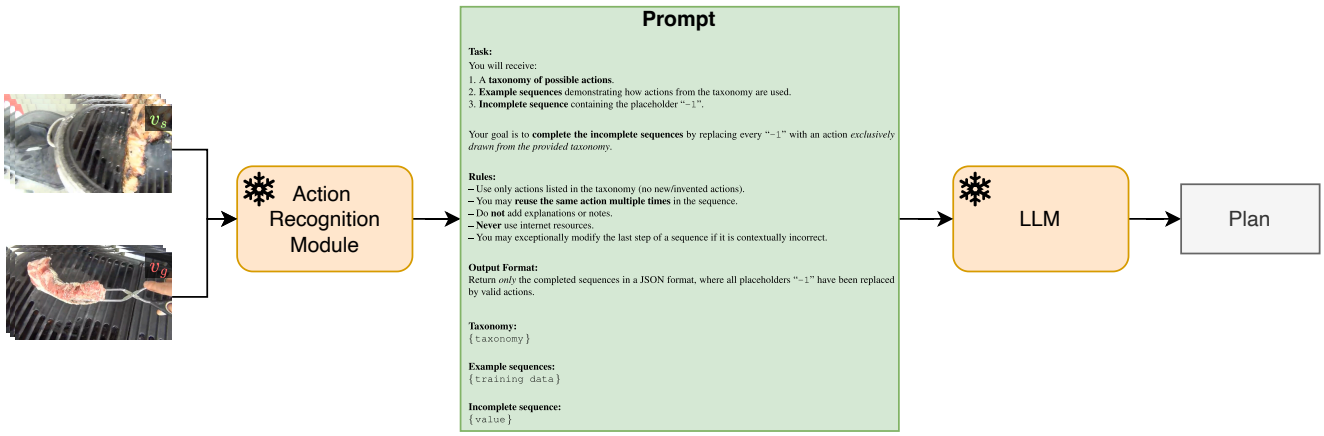

Figure 7. LLM-Based sequence completion approach.

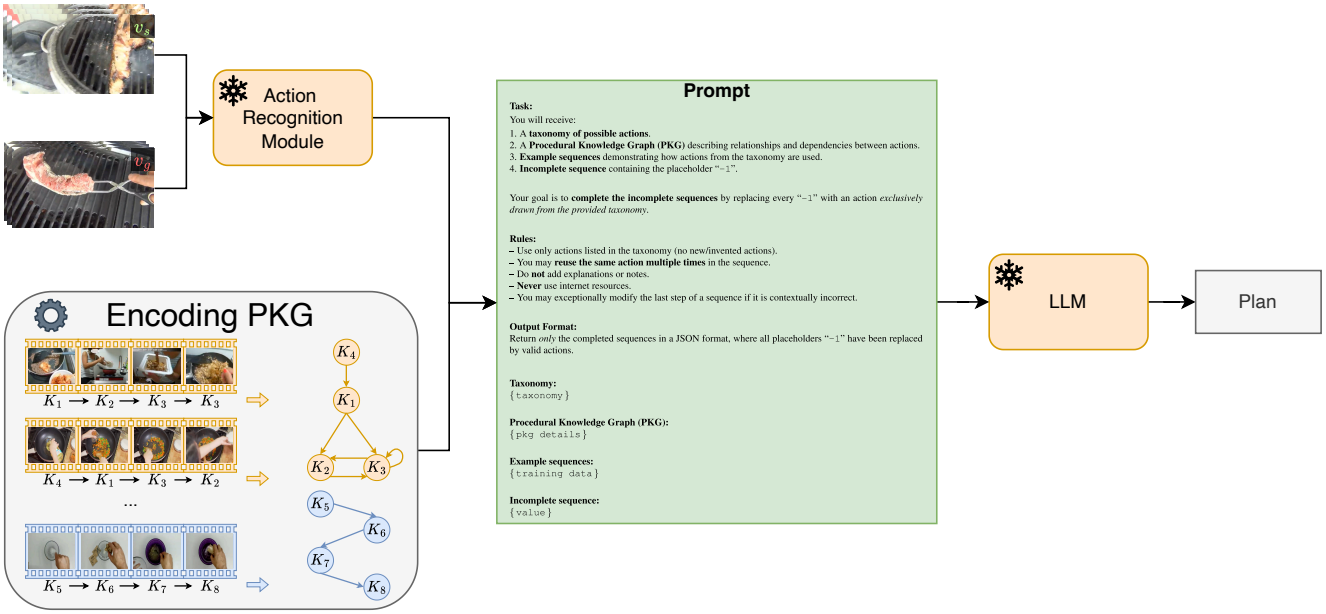

Figure 8. LLM + PKG sequence completion approach.

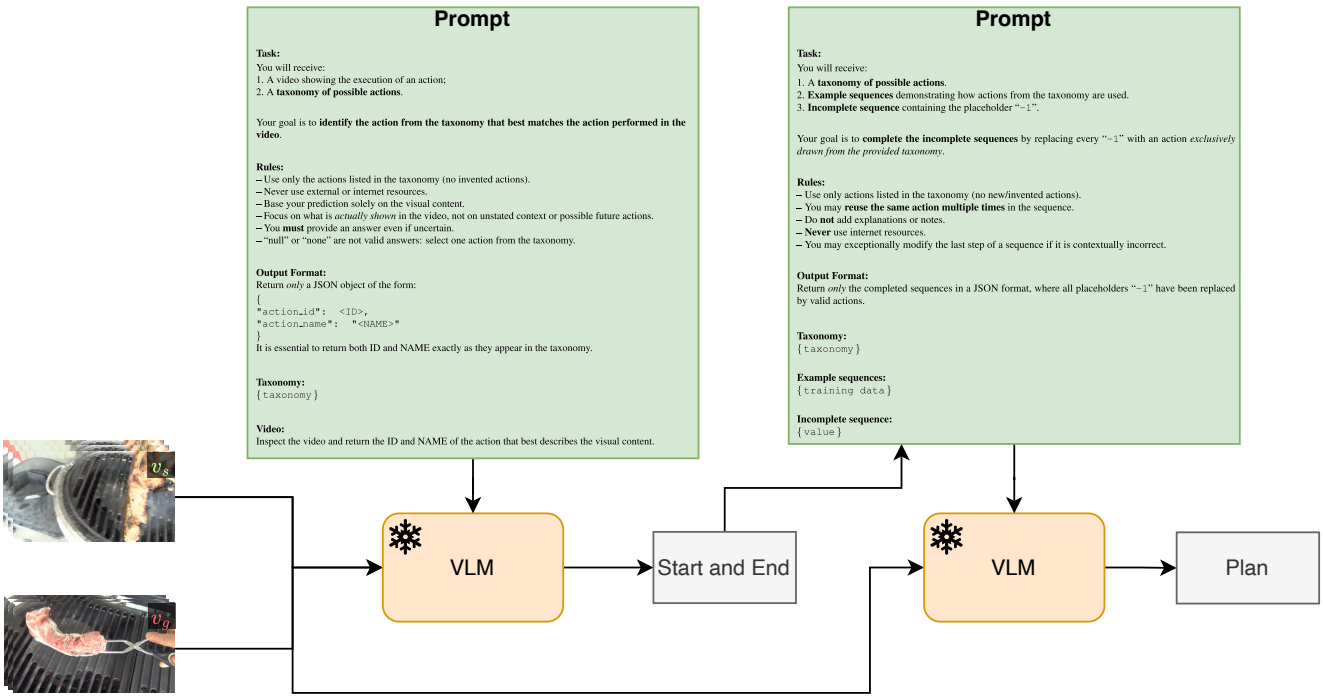

Figure 9. VLM-based sequence completion approach.

Task:
You will receive:
1. A **taxonomy of possible actions**.
2. **Example sequences** demonstrating how actions from the taxonomy are used.
3. **Incomplete sequence** containing the placeholder "`-1`".

Your goal is to **complete the incomplete sequences** by replacing every "`-1`" with an action *exclusively drawn from the provided taxonomy*.

Rules:
– Use only actions listed in the taxonomy (no new/invented actions).
– You may **reuse the same action multiple times** in the sequence.
– Do **not** add explanations or notes.
– **Never** use internet resources.
– You may exceptionally modify the last step of a sequence if it is contextually incorrect.

Output Format:
Return *only* the completed sequences in a JSON format, where all placeholders "`-1`" have been replaced by valid actions.

Taxonomy:
{`taxonomy`}

Example sequences:
{`training data`}

Incomplete sequence:
{`value`}

Prompt 1. Prompt used to instruct the model to complete action sequences from a predefined taxonomy.

Task:
You will receive:
1. A **taxonomy of possible actions**.
2. A **Procedural Knowledge Graph (PKG)** describing relationships and dependencies between actions.
3. **Example sequences** demonstrating how actions from the taxonomy are used.
4. **Incomplete sequence** containing the placeholder "`-1`".

Your goal is to **complete the incomplete sequences** by replacing every "`-1`" with an action *exclusively drawn from the provided taxonomy*.

Rules:
– Use only actions listed in the taxonomy (no new/invented actions).
– You may **reuse the same action multiple times** in the sequence.
– Do **not** add explanations or notes.
– **Never** use internet resources.
– You may exceptionally modify the last step of a sequence if it is contextually incorrect.

Output Format:
Return *only* the completed sequences in a JSON format, where all placeholders "`-1`" have been replaced by valid actions.

Taxonomy:
{`taxonomy`}

Procedural Knowledge Graph (PKG):
{`pkg details`}

Example sequences:
{`training data`}

Incomplete sequence:
{`value`}

Prompt 2. Prompt used when procedural knowledge graph (PKG) information is available for sequence completion.

Task:
You will receive:
1. A video showing the execution of an action;
2. A **taxonomy of possible actions**.

Your goal is to **identify the action from the taxonomy that best matches the action performed in the video**.

Rules:
– Use only the actions listed in the taxonomy (no invented actions).
– Never use external or internet resources.
– Base your prediction solely on the visual content.
– Focus on what is *actually shown* in the video, not on unstated context or possible future actions.
– You **must** provide an answer even if uncertain.
– "null" or "none" are not valid answers: select one action from the taxonomy.

Output Format:
Return *only* a JSON object of the form:

```
{
"action_id": <ID>,
"action_name": "<NAME>"
}
```

It is essential to return both ID and NAME exactly as they appear in the taxonomy.

Taxonomy:
{taxonomy}

Video:
Inspect the video and return the ID and NAME of the action that best describes the visual content.

Prompt 3. Prompt used for VLM-based action identification from video given a fixed action taxonomy.