GFlowVLM: Enhancing Multi-step Reasoning in Vision-Language Models with Generative Flow Networks

Supplementary Material

A. Preliminaries

A.1. GFlowNets

We summarize the necessary preliminaries of GflowNets and encourage readers to refer to [2] for deeper understanding. In a directed acyclic graph $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ with states $\mathcal{S}$ and directed actions $\mathcal{A}$, a complete trajectory is any trajectory starting in initial state s_0 and ending in terminal state $x \in X$ where $X \subset \mathcal{S}$. There is a unique initial state $s_0 \in S$ with no parents. States with no children are called terminal, and the set of terminal states is denoted by $\mathcal{X}$. A trajectory $\tau = (s_0 \to \ldots \to s_n = x)$ represents a complete sequence ending in a terminal state $x \in \mathcal{X}$ where each $(s_t \to s_{t+1})$ is an action. The trajectory flow $F : \mathcal{T} \to \mathbb{R}_{\geq 0}$ defines flows over trajectories, with state flow $F(s) = \sum_{s \in \tau} F(\tau)$ and with edge flow $F(s \to s') = \sum_{\tau=(\ldots \to s \to s' \to \ldots)} F(\tau)$. The trajectory flow F is Markovian if there exist action distributions $P_F(\cdot|s)$ over the children of each non-terminal state s.

A.1.1. Forward and Backward Policies

A forward policy $P_F(\cdot|s)$, often parametrized by a neural network, induces a distribution over trajectories and a marginal distribution over the children of every non-terminal state $s \in S$, with probabilities given by: $P_F(\tau) = P_F(s_0 \to \ldots \to s_n) = \prod_{t=0}^{n-1} P_F(s_{t+1}|s_t) \quad \forall \tau \in \mathcal{T}$. The distribution over complete trajectories that arises from a forward policy satisfies a Markov property. The forward policy can then be used to sample terminal states $x \in X$ by starting at state s_0 and iteratively sampling actions from P_F. A backward policy $P_B(\tau) = P_B(s_n \to \ldots \to s_0) = \prod_{t=0}^{n-1} P_B(s_t|s_{t+1}) \quad \forall \tau \in \mathcal{T}$. If F is markovian flow, then P_F and P_B can be computed in terms of state and edge flows as: $P_F(s'|s) = \frac{F(s \to s')}{F(s)}$ and $P_B(s|s') = \frac{F(s \to s')}{F(s')}$.
Given a non-negative reward function $R : \mathcal{X} \to \mathbb{R}_{\geq 0}$, GFlowNets aim to learn a policy such that the probability of sampling a state $x \in \mathcal{X}$ is proportional to $R(x)$. The marginal likelihood of sampling a state $x \in X$ is the sum of likelihoods of all complete trajectories that terminate at x. If the objective function is globally minimized, then the likelihood of terminating at state x is proportional to $R(x)$. Formally, the learning problem solved by GFlowNets is to estimate a policy P_F over trajectories such that there exists a normalizing constant Z satisfying: $R(x) = Z \sum_{\tau=(s_0 \to \ldots \to s_n = x)} P_F(\tau) \quad \forall x \in \mathcal{X}$, where $Z = F(s_0) = \sum_{\tau \in \mathcal{T}} F(\tau)$ is total flow at the initial state, and $\tau \in \mathcal{T}$ is the trajectory.

A.2. Motivating Example

We include a practical example from ALFWorld demonstrating how GFlowVLM can be applied to embodied AI tasks in Fig. 4. The agent is presented with a visual observation of a simulated household environment and a high-level goal in natural language, such as "Put keychain in ottoman." The task requires the agent to generate a valid sequence of actions (e.g., open drawer → take keychain from drawer → close the opened drawer → go to ottomann → place keychain in ottoman). Importantly, there are multiple valid plans that achieve the same goal, with subtle causal constraints (e.g., the drawer must be open before taking the keychain from it, and objects must be picked up before being moved). We observe that models trained with PPO tend to converge on the most common or shortest path, while GFlowVLM generates a more diverse set of valid action sequences, reflecting a richer understanding of the causal structure of the environment. This example demonstrates GFlowNets' strength in reasoning over multimodal inputs and learning structured, stochastic policies that preserve functional diversity.

B. Environments

B.1. ALFWorld

ALFWorld [31] is an embodied AI environment combining a text-based interactive setup with a vision-language planning dataset. It includes six goal-conditioned tasks: "Pick & Place", "Examine in Light", "Clean & Place", "Heat & Place", "Cool & Place", and "Pick Two & Place". The agent must plan and act based on visual cues and textual instructions (e.g., "go to shelf 1") that specify the task. Unlike gym_cards, where all states share the same action space, ALFWorld has a state-dependent action space $\mathcal{A}_t$; actions are context-dependent (e.g., "put some pillows on the armchair", the agent can only place a pillow after picking it up). Our prompt instructs the VLM to choose from the admissible actions $\mathcal{A}_t$, and we evaluate Out-Of-Distribution (OOD) performance using a test set of previously unseen scenes (see detailed prompt templates in Tab. 9 and Tab. 10). We use the same non-negative components of the reward function used in [41], which includes sub-goal and goal rewards: $r(s_t, a_t, s_{t+1}|g_{task}) = 50 * 1\{s_{t+1} = g_{task}\} + 1\{s_{t+1} = g_{task}\}$. We do not include the negative component of the reward function represented as $-1\{a_t \notin \mathcal{A}_t(s_t)\}$ in [41], since the actions are always selected from the admissible actions provided in the input prompt p_t. The rewards are non-

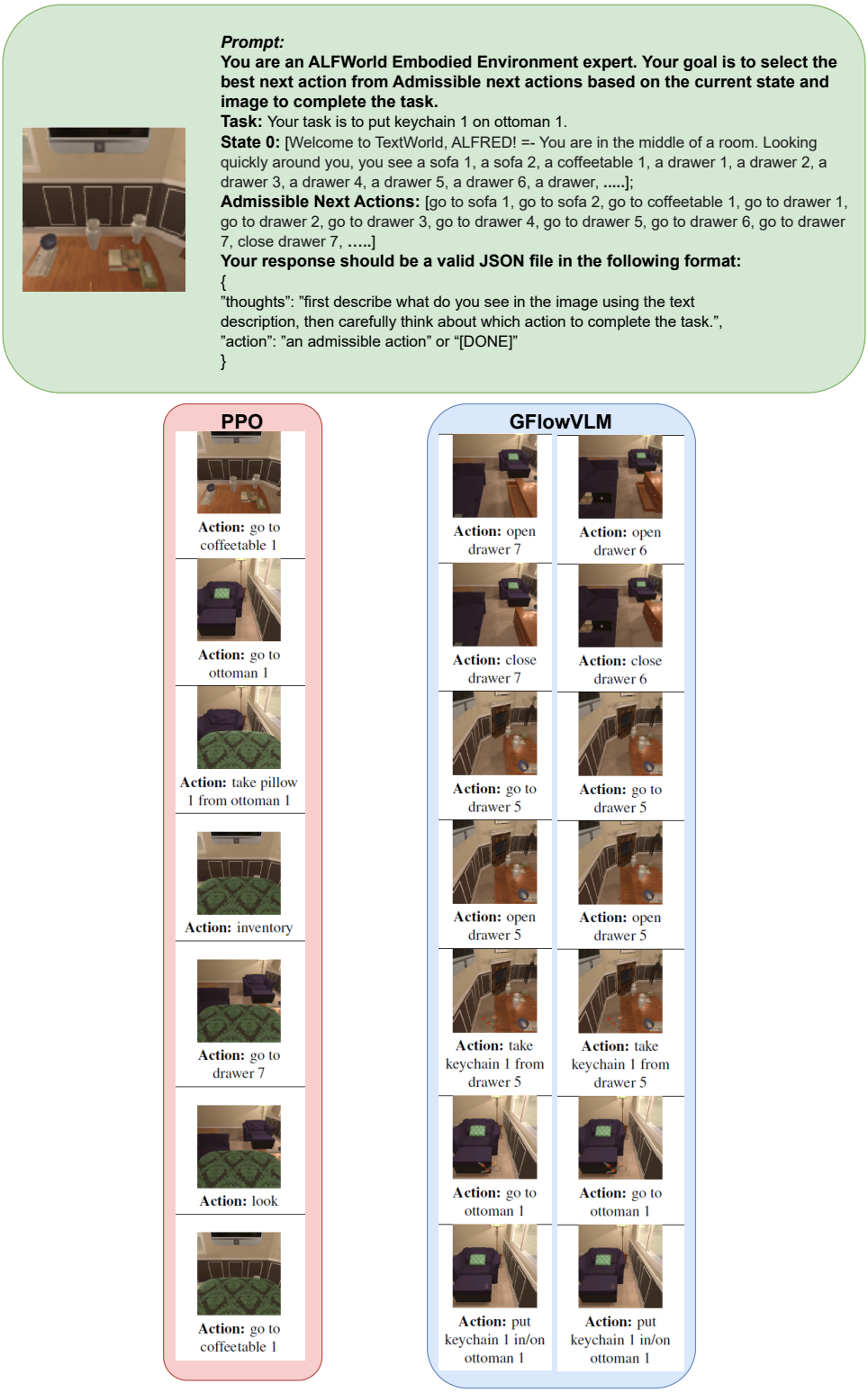

Figure 4. Overview of the prediction of diverse sequence using GFlowVLMs as compared to PPO for AlfWorld scenarios. The model takes the image of sequence and prompt as input, and generates the next number of sequence by implicitly modeling the causality.

negative, making it suitable for Var-TB and SubTB losses. However, since it lacks dense rewards for every transition, we didn't use GFlowVLM with DB loss on this task.

B.2. NumberLine

This task involves moving a number along a synthetic number line to reach a target. NumberLine requires identifying two numbers in an image: "target: c" and "current: y_t", where c and y_t are both integers such that $c, y_t \in [n_{min}, n_{max}]$. The agent's goal is to align y_t with c by outputting an action a_t from the discrete set {"+", "-", `[DONE]` (if applicable)}. Actions "+" and "-" adjust $y_t \pm 1$, while `[DONE]` signals task completion (see detailed prompt template in Tab. 7). An episode ends when the $y_t = x$, or when the maximum step $T = 2n_{max}$ is reached, which is the default setup of the environment. We set n_{min} and n_{max} as 0 and 5, respectively for the in-distribution examples, and set n_{min} and n_{max} as 10 and 50 for generating OOD examples. In the reward function used in [41], an agent receives a reward of $r(s_t, a_t) = 1$ when $y_{t+1} = c$, a penalty of $r(s_t, a_t) = -1$ upon taking an action that does not move the current number y_t to the target c, and a reward of $r(s_t, a_t) = 0$, otherwise. For GFlowVLM, we revise the reward function with non-negative values as GFlowNets inherently require non-negative as follows:

$$R(x) = R(c, y_t) = \frac{l}{|c - y_t| + 1} \tag{9}$$

where l is a scaling constant set to 100. This reward incentivizes the model to bring the current number closer to the target, progressively increasing the reward as the gap decreases. For fair comparison, we run RL4VLM [41] with revised reward structure.

B.3. Blackjack

The Blackjack task requires the VLM to reason with visual information and adapt to stochastic outcomes. The observation o_t includes two dealer cards (one face-down) and the player's cards. The agent aims to win by selecting an action a_t from {"stand", "hit", `[DONE]` (if applicable)} (see detailed prompt template in Tab. 8). In the reward function used in [41], an agent receives a reward of $r(x) = 1, 0, -1$ upon win, draw and loss, respectively. Since GFlowNets inherently require non-negative reward, we revise the reward function to replace non-negative values as follows:

$$R(x) = \max(1 \times 10^{-10}, (r(x) + 1) \times 10), \tag{10}$$

where $r(x)$ represents the environment's original reward for state x. This scales the rewards and ensures they are strictly non-negative. For fair comparison, we run RL4VLM [41] with revised reward structure.

$R(x)$ represents the desirability or quality of a complete trajectory with final state x, similar to RL. It defines the target distribution from which the GFlowNet learns to sample, where higher-reward outcomes should be sampled more frequently.

C. Training Objectives

We adopt three different objective functions of GFlowNets, *Trajectory-Balance (TB)*, *Subtrajectory-Balance (SubTB)*, and *Detailed-Balance (DB)*, to fine tune the VLM.

C.1. Variance Trajectory Balanced (Var-TB) Loss

The *Trajectory-Balanced* (TB) objective [23] ensures that the probability of generating a complete trajectory $\tau = (s_0 \to s_1 \to \cdots \to s_n = x)$ is proportional to the reward $R(\tau)$. Under the Markovian assumption, the forward policy $P_F(s_t|s_{t-1})$ transitions from state s_{t-1} to s_t, while the backward policy $P_B(s_{t-1}|s_t)$ ensures consistency between forward and backward flows. This objective is given by:

$$Z \prod_{t=1}^{n} P_F(s_t|s_{t-1}; \theta) = R(x) \prod_{t=1}^{n} P_B(s_{t-1}|s_t; \theta), \tag{11}$$

where Z is the partition function that normalizes the distribution.

We now change s to z to match our definition of state in the main paper, where $z_{0:t}$ consists of a visual observation o_t and an input prompt p_t containing goal description, history states $s_{0:t-1}$, history actions $a_{0:t-1}$, and admissible actions $\mathcal{A}_t$. We use $\top$, which is the `[DONE]` symbol, to represent the terminal state x of a trajectory. We adopt this notation because, in practice, the VLM predicts the action $\top$ to signify termination. This practical adaptation ensures consistency between the theoretical representation of terminal states and the actual predictions made by the VLM during inference.

Under the non-Markovian assumption of generating a complete trajectory $\tau = (z_0 \to z_1 \to \cdots \to z_n = x)$, and after adding goal into condition, we have:

$$Z \prod_{t=1}^{n} P_F(z_t|z_{0:t-1}, g; \theta) = R(x) \prod_{t=1}^{n} P_B(z_{t-1}|z_{t:n}, g; \theta), \tag{12}$$

From [43], an estimation Z for each trajectory τ can be expressed as:

$$\begin{aligned} \zeta(\tau; \theta, g) &= \log \frac{\prod_{t=1}^{n} P_F(z_t|z_{0:t-1}), g; \theta)}{R(x) \prod_{t=1}^{n} P_B(z_{t-1}|z_{t:n}, g; \theta)} \\ &= \log \frac{\prod_{t=1}^{n} P_F(z_t|z_{0:t-1}, g; \theta)}{R(x)} \end{aligned} \tag{13}$$

where $P_B = 1$ in our case since we formulate the trajectories as a tree structure, where a child state has only one parent state. In the optimal case, $\zeta(\tau;\theta,g)$ is equal to true $logZ$. The Variance-Trajectory-Balanced loss function aim to minimize the variance of $\zeta(\tau;\theta,g)$ across trajectories to make the balance of the trajectories. The final Variance-Trajectory-Balanced loss is then defined as:

$$\mathcal{L}_{\text{VarTB}}(\boldsymbol{\tau};\theta) = \frac{1}{K}\sum_{k=1}^{K}\left(\zeta(\tau_k;\theta,g) - \mathbb{E}_{\boldsymbol{\tau}}\left[\zeta(\boldsymbol{\tau};\theta,g)\right]\right)^2, \tag{14}$$

where K represents the number of sampled trajectories. This loss ensures that high-reward trajectories are sampled more frequently by the policy.

C.2. Subtrajectory Balanced (SubTB) Loss

The *Subtrajectory-Balanced* (SubTB) loss [22] operates on subtrajectories of the form $\tau = (z_0 \to z_1 \to \cdots \to z_m)$. The subtrajectory balance ensures that each segment of the reasoning path or structure remains consistent, where the flows are balanced locally between forward and backward transitions. Under the non-Markovian assumption and after adding goal into conditions, the subtrajectory balance condition is expressed as:

$$\begin{aligned} F(z_0)\prod_{t=1}^{m} P_F(z_t|z_{0:t-1}),g;\theta) = \\ F(z_m)\prod_{t=1}^{m} P_B(z_{t-1}|z_{t:m}),g;\theta), \end{aligned} \tag{15}$$

where $F(z_0)$ and $F(z_m)$ represent the flow into the initial (z_0) and final state (z_m) of the subtrajectory, respectively. Following [27], when all states z_t are terminable with $\top$, we have $F(z_t)P_F(\top|z_{0:t}) = R(\top)$. Then the SubTB loss can be formulated as:

$$\mathcal{L}_{\text{SubTB}}(z_{0:m},g;\theta) = \sum_{0\le i<j\le m} \left(\log \frac{R(z_{0:i}\top)\prod_{k=i+1}^{j} P_F(z_k|z_{0:k-1},g;\theta)P_F(\top|z_{0:j},g;\theta)}{R(z_{0:j}\top)P_F(\top|z_{0:i},g;\theta)}\right)^2 \tag{16}$$

where $\top$ is the `[DONE]` symbol, denoting a trivial terminal state, and process continues until `[DONE]` symbol $\top$ is generated similar to [8]. This loss penalizes discrepancies in local transitions and ensures that all subsegments of a trajectory follow the correct balance conditions, reducing variance in smaller parts of the trajectory.

C.3. Detailed Balanced (DB) Loss

The *Detailed-Balanced* (DB) loss [2] is used to ensure that each transition $s_t \to s_{t+1}$ between two states is balanced by matching the forward and backward flows at every step of the trajectory. The detailed balance condition is expressed as:

$$F(s_t)P_F(s_{t+1}|s_t) = F(s_{t+1})P_B(s_t|s_{t+1}), \tag{17}$$

where $F(s_t)$ and $F(s_{t+1})$ represent the flow at states s_t and s_{t+1}, respectively. Under the non-Markovian assumption of generating a complete trajectory $\tau = (z_0 \to z_1 \to \cdots \to z_n \to \top)$, where $\top$ is the terminal state of the sequence, DB loss is formulated as:

$$\mathcal{L}_{\text{DB}}(z_{0:t} \to z_{0:t+1},g;\theta) = \left(\log \frac{R(z_{0:t}\top)P_F(z_{t+1}|z_{0:t},g;\theta)P_F(\top|z_{0:t+1},g;\theta)}{R(z_{0:t+1}\top)P_F(\top|z_{0:t},g;\theta)}\right)^2. \tag{18}$$

This loss ensures that every state-to-state transition follows the correct flow, preventing inconsistencies in the trajectory construction.

Comparisons of Loss Functions TB loss controls the variance of ζ for the sampled trajectories, not the individual trajectory. Its main role is to bias sampling so that trajectory selection probability aligns with rewards [9]. In addition, DB loss excels with dense rewards by ensuring flow consistency at each state, while SubTB and TB perform better in sparse settings by optimizing flow across (sub)trajectories. Additionally, TB is suited for tasks with known full sequences, and SubTB for costly large-trajectory sampling.

Computational Complexity In practice, we calculate (sub)trajectory or transition-based loss functions, which operate over (sub)trajectories or sampled transitions rather than the full state space. This allows us to efficiently handle the non-Markovian dependencies with *linear* complexity.

D. Details of Experimental Setup

In this section, we outline the experimental setup used to evaluate our approach across various tasks. We describe the key components of our implementation, including the data collection, diversity metric, and hyperparameters. By providing these details, we aim to ensure reproducibility and clarify how the proposed method integrates into different experimental frameworks.

D.1. Off-Policy Data Collection

In this section, we describe our approach to off-policy data collection used in GFlowVLM for two distinct tasks, Numberline and Blackjack, emphasizing the integration of high-quality trajectories to enhance model training. These strategies ensure that the model learns from both successful and diverse trajectories, even when its on-policy performance falls short.

Numberline During training, if the on-policy trajectory generated by the model fails to move the current number correctly towards the target, we augment the dataset by adding an off-policy, ground-truth trajectory to the buffer. These ground-truth trajectories represent successful paths that the model can follow to achieve the goal. By incorporating these accurate trajectories, we provide the model with additional supervision, which helps it learn to generalize better to unseen instances. This ensures the model benefits from examples of correct behavior, even when its predictions deviate from the optimal path. Fig. 6 illustrates the generation of both correct and incorrect trajectories, highlighting how diversity in training trajectories is encouraged to improve robustness.

Blackjack For the stochastic Blackjack task, deterministic ground-truth trajectories are not directly available due to the probabilistic outcomes of card draws. Instead, we generate high-quality off-policy trajectories using a *rule-based heuristic*: The agent "stands" when the hand value is 17 or higher and "hits" otherwise. This strategy aligns with fundamental Blackjack principles, balancing the risk of exceeding a hand value of 21 against the potential for improvement by drawing additional cards. By leveraging this rule-based approach, we ensure that the training buffer includes trajectories that reflect a realistic yet principled decision-making process. Figure 7 demonstrates how both correct and incorrect trajectories are generated in a tree structure, promoting diversity in the training data and enabling the model to better handle a range of scenarios.

D.2. SFT Dataset Collection

To create the SFT dataset, we iteratively interact with the environment to generate successful trajectories. For each successful trajectory, we manually append the "`[DONE]`" token as the final action in the last state, explicitly marking the completion of the task. This approach aims to teach the model to predict the "`[DONE]`" token as the appropriate action when the goal state is achieved.

Numberline For the Numberline task, we execute ground-truth actions in the environment until the current state matches the target state. At this point, we append the "`[DONE]`" token to indicate task completion. This process generated 8,000 data points with "`[DONE]`" actions and 20,000 additional data points for other actions, using the base SFT dataset in [41].

Blackjack For Blackjack, we adhere to the standard 17-point rule to determine actions. When the optimal decision is to take no further action, we append the "`[DONE]`" token to the trajectory. This yielded 15,000 data points with "`[DONE]`" actions and 50,000 for other actions, utilizing the SFT dataset from [41].

ALFWorld For ALFWorld, we rely on expert actions derived from a heuristic [31]. At the end of each successful trajectory, we append the "`[DONE]`" token to signify task completion. This resulted in 15,000 data points with "`[DONE]`" actions and 45,000 for other actions using the SFT dataset from [41].

D.3. Diversity Metric

The diversity metric introduced in [40] calculates the diversity of successful trajectories found by a policy under the same number of samplings at inference time. Specifically, it is defined as follows:

$$Div = \frac{\sum_{i=1}^{n} S_i \cdot \mathbb{I}(S_i \geq 1)}{\sum_{i=1}^{n} \mathbb{I}(S_i \geq 1)} \geq 1 \tag{19}$$

where n is the total number of tasks, S_i is the number of successful trajectories found for the i-th task, and $\mathbb{I}(S_i \geq 1)$ is an indicator function that equals 1 if at least one successful trajectory is found for the i-th task, and 0 otherwise. The denominator represents the number of tasks where the model finds at least one successful trajectory, while the numerator sums the total number of successful trajectories across all tasks. The smallest possible *Div* is 1, indicating that a method finds at least one successful trajectory on average. For example, a $Div = 1.2$ suggests that, on average, a method finds 1.2 different successful trajectories. The (*Div*@N) metric used in the main paper represents the diversity of successful trajectories after sampling N trajectories' samples.

D.4. General Setup for Baselines and GFlowVLM

All experiments are conducted on an H100 DGX machine with 80GB of memory. During VLM training, we directly optimize all trainable components, including the vision encoder, LLM, and MLP projector. For baseline methods, we utilize the open-source implementations provided in the original papers for SFT and RL4VLM [41]. A *CosineAnnealingLR* scheduler is adopted, starting with an initial learning rate of 1×10^{-5}, decaying to a final learning rate of 1×10^{-9}, and reaching its maximum learning rate at step

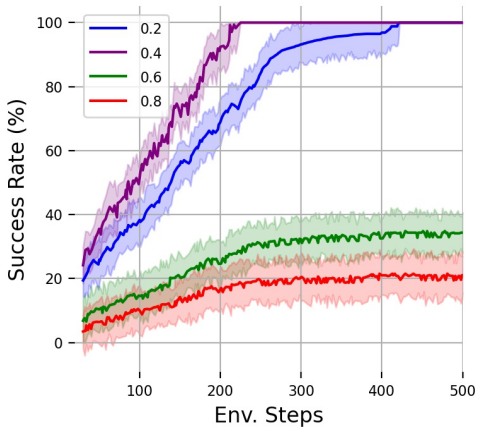

Figure 5. Average success rates (%) of our method under different CoT weighting factor λ on NumberLine across three loss functions.

25. For GFlowVLM, a buffer size of 4 is used across all tasks. To ensure a fair comparison, we report the number of environment steps for each method.

D.5. CoT Weighting Factor λ

$$P_F(z_{t+1}|z_{0:t}, g; \theta) = P_{\text{Action}}(a_t|z_{0:t}, c_t, g; \theta) + \lambda P_{\text{CoT}}(c_t|z_{0:t}, g; \theta), \quad (20)$$

The CoT weighting factor, $\lambda \in [0, 1]$, controls the influence of CoT reasoning within our framework, as discussed briefly in the main paper (rewritten here in Eq. (20)). To assess the impact of λ, we compute the average performance of our proposed framework, GFlowVLM, using three loss functions, each evaluated with four random seeds. As shown in Figure 5, a moderate λ (e.g., 0.4) yields the best performance on NumberLine tasks across three different loss functions. When λ is too high (0.8) or too low (0.2), $P_{\text{CoT}}(c_t|z_{0:t}, g; \theta)$ or $P_{\text{Action}}(a_t|z_{0:t}, c_t, g; \theta)$ overly influences the estimation of P_F, respectively, leading to imbalanced learning dynamics. Thus, setting $\lambda = 0.4$ effectively balances CoT and action learning, enhancing reasoning performance. We use the same value of $\lambda = 0.4$ across all experiments in this work.

E. Qualitative Results

We present an example in ALFWorld in Tab. 11, with the goal of "put some keychains on the ottoman" to illustrate key insights into our method.

Our method encourages exploration by sampling proportional to the reward, allowing it to avoid getting stuck in suboptimal states—a common limitation observed in PPO. This exploration not only prevents suboptimal convergence

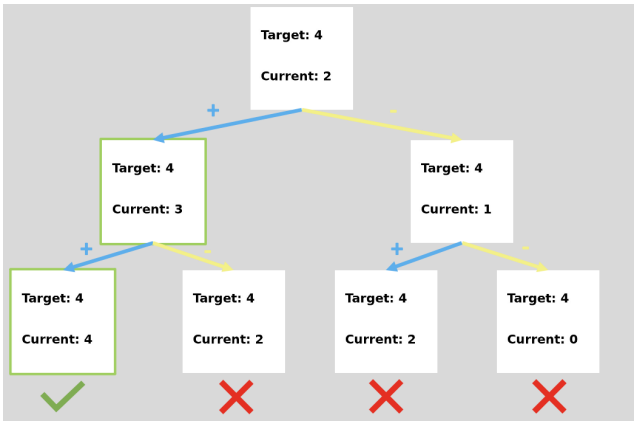

Figure 6. An example of off-policy data collection for NumberLine in a tree structure.

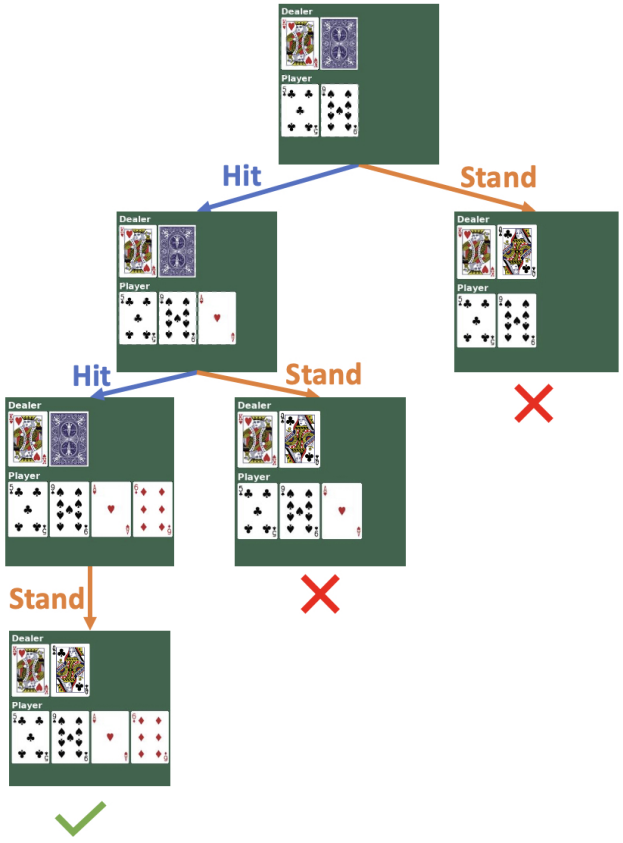

Figure 7. An example of off-policy data collection for BlackJack in a tree structure.

but also enables the model to generate more diverse solutions, as demonstrated by the multiple trajectories shown in Tab. 11. Through repeated sampling, our method effectively considers a wider range of potential paths to achieve the goal.

PPO, in contrast, tends to rely on superficial semantic

Method	Train Data	Assump.	SFT Init.	NL	NL-OOD	BJ
Ablations of RL4VLM						
RL4VLM [41]*	On	M	✓	34.8	1.9	23.5
RL4VLM [41]	On	M	✓	89.4	3.1	40.2
RL4VLM [41]	On	NM	✓	**90.3**	**4.4**	**41.0**
Ablations of GFlowVLM w/ Var-TB w/ On and Off-Policy						
GFlowVLM w/ Var-TB	On	M	✓	93.4	4.7	41.0
GFlowVLM w/ Var-TB	On	NM	✓	**100.0**	6.2	41.4
GFlowVLM w/ Var-TB	Off	M	✓	94.5	17.2	42.0
GFlowVLM w/ Var-TB	Off	NM	✓	**100.0**	**17.3**	**43.0**
Ablations of GFlowVLM w/ SubTB w/ On and Off-Policy						
GFlowVLM w/ SubTB	On	M	✓	91.7	4.0	40.2
GFlowVLM w/ SubTB	On	NM	✓	**100.0**	7.0	41.7
GFlowVLM w/ SubTB	Off	M	✓	94.8	**17.3**	40.5
GFlowVLM w/ SubTB	Off	NM	✓	**100.0**	16.7	**42.4**
Ablations of GFlowVLM w/ DB w/ On and Off-Policy						
GFlowVLM w/ DB	On	M	✓	90.1	5.3	40.0
GFlowVLM w/ DB	On	NM	✓	**100.0**	9.1	42.2
GFlowVLM w/ DB	Off	M	✓	93.6	16.3	41.5
GFlowVLM w/ DB	Off	NM	✓	**100.0**	**18.6**	**43.8**

Table 5. Ablations of GFlowVLM with Markovian assumption for NumberLine (NL) and BlackJack (BJ) tasks for in-distribution and out-of-distributions (OOD) tasks. *We use the same reward function as ours. NL-OOD stands for Number line with out-of-distribution tasks. On and Off represent On-Policy and Off-Policy, respectively. M and NM stands for Markovian and non-Markovian assumption respectively.

Method	Assump.	Pick	Look	Clean	Heat	Cool	Pick2	Avg.	OOD	Div@16
Ablations of RL4VLM										
RL4VLM [41]	M	47.4	**14.7**	**10.4**	14.4	18.8	18.0	21.7	4.8	**1.12**
RL4VLM [41]	NM	**49.1**	13.5	9.8	**15.2**	**20.1**	**20.6**	**22.1**	**6.1**	1.11
Ablations of GFlowVLM w/ SubTB										
GFlowVLM w/ SubTB	M	46.0	10.1	9.7	14.7	**24.6**	23.7	22.1	8.0	1.34
GFlowVLM w/ SubTB	NM	**50.0**	**23.1**	**10.0**	**18.7**	24.3	23.7	**26.1**	**12.3**	**1.40**
Ablations of GFlowVLM w/ Var-TB										
GFlowVLM w/ Var-TB	M	45.1	12.2	**11.3**	**15.7**	20.6	**24.7**	22.9	7.6	1.37
GFlowVLM w/ Var-TB	NM	**50.0**	**22.2**	10.2	16.1	**22.7**	21.9	**25.7**	**10.9**	**1.41**

Table 6. Ablations of GFlowVLM with Markovian assumption for ALFWorld. Since Alfworld does not provide dense rewards, we can not not using DB loss here. Furthermore, while RL4VLM and GFlowVLM with SubTB are trained with SFT initialization, GFVLM with TB-Var is without STF initialization since we do not need to model the flow. M and NM stands for Markovian and non-Markovian assumption respectively.

patterns to make decisions. For instance, it may prioritize reaching the "ottoman" directly without first retrieving the keychains, as the term "ottoman" semantically aligns with the goal. This behavior highlights the risk of overfitting to pattern recognition rather than aligning actions with the ultimate reward.

Image input:

NumberLine prompt template without history information (Markovian)

You are playing a game called number line. You will see a target number and a current number in the image. And your goal is to move the current number closer to the target by choosing either adding or subtracting one to the current number. You need to first give the thoughts and then you can choose between `["+", "-"]`. Use "`[DONE]`" when you think you have completed the task. Your response should be a valid JSON file in the following format:

{

"current number": "x",

"target number": "x",

"thoughts": "first read out the current and target number, then think

carefully about which action to choose",

"action": "-" or "+" or "`[DONE]`"

}

NumberLine prompt template with history information (non-Markovian)

You are playing a game called number line. You will see a target number and a current number in the image. And your goal is to move the current number closer to the target by choosing either adding or subtracting one to the current number. Below are the history actions and states you've done.

State 0: 1

Action 1: `"+"`

State 1: 2

Based on the history information, you need to first give the thoughts and then you can choose between `["+", "-"]`. Use "`[DONE]`" when you think you have completed the task.Your response should be a valid JSON file in the following format:

{

"current number": "x",

"target number": "x",

"thoughts": "first read out the current and target number, then think

carefully about which action to choose",

"action": "-" or "+"or "`[DONE]`"

}

Table 7. Prompt Template with Markovian and non-Markovian assump. for NumberLine. The sentence in brown is only applicable for SubTB and DB losses.

Image input:

BlackJack prompt template without history information (Markovian)

You are a blackjack player. You are observing the current game state. You need to first give an explanation and then you can choose between `["stand", "hit"]`. Use "`[DONE]`" when you think you have completed the task. Your response should be a valid JSON file in the following format:

{

"thoughts": "first describe your total points and the dealer's total points then think about which action to choose",
"action": "stand" or "hit" or "`[DONE]`"

}

BlackJack prompt template with history information (non-Markovian)

You are a blackjack player. You are observing the current game state. Below are the history actions and states.

State 0: 14 points

Action 1: `"hit"`

State 1: 15 points

Based on the history information, you need to first give an explanation and then you can choose between `[``stand", ``hit"]`. Use "`[DONE]`" when you think you have completed the task. Your response should be a valid JSON file in the following format:

{

"thoughts": "first describe your total points and the dealer's total points then think about which action to choose",
"action": "stand" or "hit" or "`[DONE]`"

}

Table 8. Prompt Templates with Markovian and non-Markovian assump. for BlackJack. The sentence in brown is only applicable for SubTB and DB losses.

Image input:

ALFWorld prompt template without history information (Markovian)

You are an ALFWorld Embodied Environment expert. Your goal is to select the best next action from the Admissible Next Actions based on the current state and image to complete the task. Use "`[DONE]`" when you think you have completed the task.

Task: Your task is to put a cool mug in cabinet.

Current State: `"['You arrive at loc 1. The cabinet 1 is open. On the cabinet 1, you see a pan 1, a kettle 1, a winebottle 1, a apple 1, a stoveknob 1, a stoveknob 2, a stoveknob 3, a stoveknob 4, a knife 1, a saltshaker 1, and a bread 1.']."`

Admissible Next Actions: `['go to countertop 1', 'go to cabinet 2', 'go to countertop 2', 'go to stoveburner 1', 'go to drawer 1', 'go to drawer 2', 'go to drawer 3', 'go to stoveburner 2', 'go to stoveburner 3', 'go to stoveburner 4', 'go to drawer 4', 'go to cabinet 3', 'go to cabinet 4', 'go to microwave 1', 'go to cabinet 5', 'go to cabinet 6', 'go to cabinet 7', 'go to sink 1', 'go to sinkbasin 1', 'go to fridge 1', 'go to toaster 1', 'go to coffeemachine 1', 'go to cabinet 8', 'go to drawer 5', 'go to drawer 6', 'go to drawer 7', 'go to drawer 8', 'go to shelf 1', 'go to shelf 2', 'go to countertop 3', 'go to shelf 3', 'go to drawer 9', 'go to garbagecan 1', 'open cabinet 1', 'close cabinet 1', 'take pan 1 from cabinet 1', 'take kettle 1 from cabinet 1', 'take winebottle 1 from cabinet 1', 'take apple 1 from cabinet 1', 'take stoveknob 1 from cabinet 1', 'take stoveknob 2 from cabinet 1', 'take stoveknob 3 from cabinet 1', 'take stoveknob 4 from cabinet 1', 'take knife 1 from cabinet 1', 'take saltshaker 1 from cabinet 1', 'take bread 1 from cabinet 1', 'inventory', 'look', 'examine cabinet 1'].`

Your response should be a valid JSON file in the following format:

{

"thoughts": "first describe what do you see in the image using the text description, then carefully think about which action to complete the task.",

"action": "an admissible action" or "`[DONE]`"

}

Table 9. Prompt template with Markovian assump. for ALFWorld. The sentence in brown is only applicable for SubTB and DB losses.

Image input:

ALFWorld prompt template with history information (non-Markovian)

You are an ALFWorld Embodied Environment expert. Your goal is to select the best next action from the Admissible Next Actions based on the previous and current states and image to complete the task. Use "`[DONE]`" when you think you have completed the task.

Task: Your task is to put a cool mug in cabinet.

State 0: `['-= Welcome to TextWorld, ALFRED! =- You are in the middle of a room. Looking quickly around you, you see a countertop 1, a coffeemachine 1, a cabinet 1, a cabinet 2, a cabinet 3, a sink 1, a cabinet 4, a drawer 1, a drawer 2, a drawer 3, a sinkbasin 1, a cabinet 5, a toaster 1, a fridge 1, a cabinet 6, a cabinet 7, a cabinet 8, a microwave 1, a cabinet 9, a cabinet 10, a cabinet 11, a drawer 4, a cabinet 12, a drawer 5, a stoveburner 1, and a stoveburner 2.']`
Action 1: `"open cabinet 1."`
State 1: `"['You arrive at loc 1. The cabinet 1 is open. On the cabinet 1, you see a pan 1, a kettle 1, a winebottle 1, a apple 1, a stoveknob 1, a stoveknob 2, a stoveknob 3, a stoveknob 4, a knife 1, a saltshaker 1, and a bread 1.']."`

Admissible Next Actions: `['go to countertop 1', 'go to cabinet 2', 'go to countertop 2', 'go to stoveburner 1', 'go to drawer 1', 'go to drawer 2', 'go to drawer 3', 'go to stoveburner 2', 'go to stoveburner 3', 'go to stoveburner 4', 'go to drawer 4', 'go to cabinet 3', 'go to cabinet 4', 'go to microwave 1', 'go to cabinet 5', 'go to cabinet 6', 'go to cabinet 7', 'go to sink 1', 'go to sinkbasin 1', 'go to fridge 1', 'go to toaster 1', 'go to coffeemachine 1', 'go to cabinet 8', 'go to drawer 5', 'go to drawer 6', 'go to drawer 7', 'go to drawer 8', 'go to shelf 1', 'go to shelf 2', 'go to countertop 3', 'go to shelf 3', 'go to drawer 9', 'go to garbagecan 1', 'open cabinet 1', 'close cabinet 1', 'take pan 1 from cabinet 1', 'take kettle 1 from cabinet 1', 'take winebottle 1 from cabinet 1', 'take apple 1 from cabinet 1', 'take stoveknob 1 from cabinet 1', 'take stoveknob 2 from cabinet 1', 'take stoveknob 3 from cabinet 1', 'take stoveknob 4 from cabinet 1', 'take knife 1 from cabinet 1', 'take saltshaker 1 from cabinet 1', 'take bread 1 from cabinet 1', 'inventory', 'look', 'examine cabinet 1'].`

Your response should be a valid JSON file in the following format:

{

"thoughts": "first describe what do you see in the image using the text description, then carefully think about which action to complete the task.",

"action": "an admissible action" or "`[DONE]`"

}

Table 10. Prompt template with non-Markovian assump. for ALFWorld. The sentence in brown is only applicable for SubTB and DB losses.

Goal: put some keychains on ottoman.		
PPO	**Ours-Traj. 1**	**Ours-Traj. 2**
 **Action:** go to coffeetable 1	 **Action:** open drawer 6	 **Action:** open drawer 7
 **Action:** go to ottoman 1	 **Action:** close drawer 6	 **Action:** close drawer 7
 **Action:** take pillow 1 from ottoman 1	 **Action:** go to drawer 5	 **Action:** go to drawer 5
 **Action:** inventory	 **Action:** open drawer 5	 **Action:** open drawer 5
 **Action:** go to drawer 7	 **Action:** take keychain 1 from drawer 5	 **Action:** take keychain 1 from drawer 5
 **Action:** look	 **Action:** go to ottoman 1	 **Action:** go to ottoman 1
 **Action:** go to coffeetable 1	 **Action:** put keychain 1 in/on ottoman 1	 **Action:** put keychain 1 in/on ottoman 1

Table 11. Qualitative results for ALFWorld task. GFlowVLM generates diverse trajectories in contrast to PPO.

F. Ablation Study of Markovian and non-Markovian

To evaluate the impact of Markovian and non-Markovian assumptions on performance, we conduct an ablation study with our method, GFlowVLM with both On-Policy and Off-Policy training, and RL4VLM [41] across 3 tasks: NumberLine and Blackjack and ALFWorld. The primary difference between these two assumptions lies in the prompt template used during training. Under the Markovian assumption, the model operates with prompts that do not include historical information about prior actions and states, relying solely on the current state. Conversely, the non-Markovian assumption incorporates the history of actions and states into the prompt, providing richer contextual information (see prompt templates in Tab. 7, Tab. 8, Tab. 10, Tab. 9 for details).

As shown in Tab. 5, the non-Markovian assumption leads to consistently better performance across all tasks. In NumberLine and Blackjack, GFlowVLM achieves substantial improvements in both in-distribution and out-of-distribution scenarios under the non-Markovian assumption.For instance, in the Numberline task, GFlowVLM with the DB loss demonstrates improved out-of-distribution performance when transitioning from Markovian to non-Markovian assumptions. Specifically, with on-policy training, the performance increases from 5.3 to 9.1, while with off-policy training, it rises from 16.3 to 18.6. Similarly, in Blackjack, non-Markovian prompts result in a higher average success rate.

In ALFWorld tasks, as demonstrated in Tab. 6, the non-Markovian assumption yields marked gains in both average performance and out-of-distribution generalization. For instance, GFlowVLM with SubTB achieves an average success rate of 26.1 under the non-Markovian assumption compared to 22.1 under the Markovian setup. These results highlight the importance of historical context in improving task performance, particularly for challenging scenarios requiring long-term dependencies.

Interestingly, the non-Markovian assumption also benefits the baselines, including RL4VLM, resulting in a performance increase from 3.1 to 4.4 for Numberline for OOD tasks. This suggests that GFlowVLM is better equipped to leverage the additional context provided by non-Markovian prompts, enabling it to capture richer dependencies and improve both accuracy and diversity. Overall, the findings confirm that the non-Markovian assumption provides a more effective framework for reasoning-based tasks, particularly when combined with GFlowVLM's structured learning approach.