FAST INFERENCE AND TRANSFER OF COMPOSITIONAL TASK FOR FEW-SHOT TASK GENERALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We tackle real-world problems with complex structures beyond the pixel-based game or simulator. We formulate it as a few-shot reinforcement learning problem where a task is characterized by a subtask graph that defines a set of subtasks and their dependencies that are unknown to the agent. Different from the previous meta-rl methods trying to directly infer the unstructured task embedding, our multi-task subtask graph inferencer (MTSGI) first infers the common high-level task structure in terms of the subtask graph from the training tasks, and use it as a prior to improve the task inference in testing. Our experiment results on 2D grid-world and complex web navigation domains show that the proposed method can learn and leverage the common underlying structure of the tasks for faster adaptation to the unseen tasks than various existing algorithms such as meta reinforcement learning, hierarchical reinforcement learning, and other heuristic agents.

1 INTRODUCTION

Recently, deep reinforcement learning (RL) has shown an outstanding performance on various domains such as video games (Mnih et al., 2015; Vinyals et al., 2019) and board games (Silver et al., 2017). However, most of the successes of deep RL were focused on a single-task setting where the agent is allowed to interact with the environment for hundreds of millions of time steps. In numerous real-world scenarios, interacting with the environment is expensive or limited, and the agent is often presented with a novel task that is not seen during its training time. To overcome this limitation, many recent works focused on scaling the RL algorithm beyond the single-task setting. Recent works on multi-task RL aim to build a single, contextual policy that can solve multiple related tasks and generalize to unseen tasks. However, they require a certain form of task embedding as an extra input that often fully characterizes the given task (Oh et al., 2017; Andreas et al., 2017; Yu et al., 2017; Chaplot et al., 2018), or requires a human demonstration Huang et al. (2018), which are not readily available in practice. Meta RL (Finn et al., 2017; Duan et al., 2016) focuses on a more general setting where the agent should learn about the unseen task purely via interacting with the environment without any additional information. However, such meta-RL algorithms either require a large amount of experience on the diverse set of tasks or are limited to a relatively smaller set of simple tasks with a simple task structure.

On the contrary, real-world problems require the agent to solve much more complex and compositional tasks without human supervision. Consider a web-navigating RL agent given the task of checking out the products from an online store as shown in Figure 1. The agent can complete the task by filling out the required web elements with the correct information such as shipping or payment information, navigating between the web pages, and placing the order. Note that the task consists of multiple *subtasks* and the subtasks have complex dependencies in the form of *precondition*; for instance, the agent may proceed to the payment web page (see *Bottom*, B) *after* all the required shipping information has been correctly filled in (see *Bottom*, A), or the `credit_card_number` field will appear *after* selecting the `credit_card` as a payment method (see *Top, Middle* in Figure 1). Learning to perform such a task can be quite challenging if the reward is given only after yielding meaningful outcomes (*i.e.*, sparse reward task). This is the problem scope we focus on in this work: solving and generalizing to unseen compositional sparse-reward tasks with complex subtask dependencies without human supervision.

Recent works (Sohn et al., 2019; Xu et al., 2017; Huang et al., 2018; Liu et al., 2016; Ghazanfari & Taylor, 2017) tackled the compositional tasks by explicitly inferring the underlying task structure

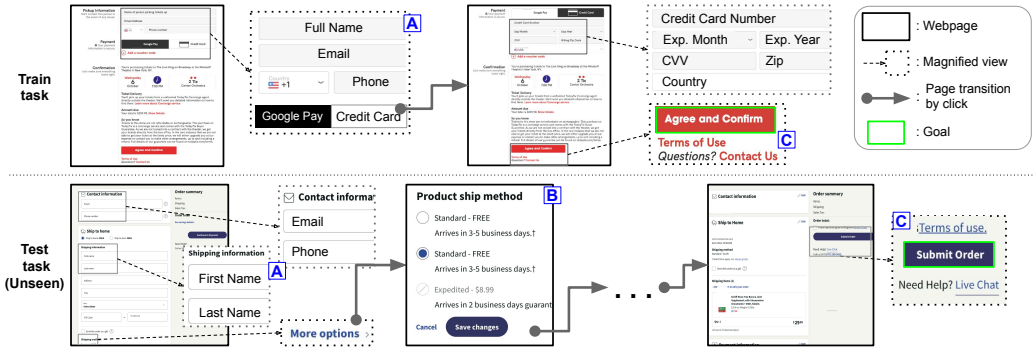

Figure 1: An illustration of the train *(Top)*and test task *(Bottom)*in our *SymWoB* domain. Some selected actionable web-elements (*e.g.*, text fields and buttons) are magnified (dotted arrow and box) for readability. The agent's goal (green box) is to checkout the products in unseen test website by interacting with the web elements in a correct order. For example, in train task, the agent should fill out all the text fields in (*Top*, **A**) **before** clicking the `credit_card` button to transition (gray arrow) to next page. The high-level checkout processes in different websites have many commonalities while certain details may differ. For example, in both train and test tasks, the agent should fill out the user information (*Top* and *Bottom*, **A**) before proceeding to the next page or there exist similar elements (*Top* and *Bottom*, **C**). However, the details may differ; *e.g.*, the train task (*Top*, **A**) has a single text field for full name, while the test task (*Bottom*, **A**) has separate text fields for the first and last name, respectively. Also, only the test website (*Bottom*, **B**) requires shipping information since the training website does not ship the product.

in a graph form. Specifically, the subtask graph inference (SGI) framework (Sohn et al., 2019) uses inductive logic programming (ILP) on the agent's own experience to infer the task structure in terms of *subtask graph* and learns a contextual policy to *execute* the inferred task in few-shot RL setting. However, it only meta-learned the adaptation policy that relates to the efficient exploration, while the task inference and execution policy learning were limited to a single task (*i.e.*, both task inference and policy learning were done from scratch for each task), limiting its capability of handling large variance in the task structure. We claim that the inefficient task inference may hinder applying the SGI framework to a more complex domain such as web navigation (Shi et al., 2017; Liu et al., 2018) where a task may have a large number of subtasks and complex dependencies between them. We note that humans can navigate an unseen website by transferring the high-level process learned from previously seen websites.

Inspired by this, we extend the SGI framework to a *multi-task subtask graph inferencer* (MTSGI) that can generalize the previously learned task structure to the unseen task for faster adaptation and stronger generalization. Figure 2 outlines our method. MTSGI estimates the prior model of the subtask graphs from the training tasks. When an unseen task is presented, MTSGI samples the prior that best matches with the current task, and incorporates the sampled prior model to improve the latent subtask graph inference, which in turn improves the performance of the evaluation policy. We demonstrate results in the 2D grid-world domain and the web navigation domain that simulates the interaction with 20 actual websites. We compare our method with MSGI (Sohn et al., 2019) that learns the task hierarchy from scratch for each task, and two other baselines including hierarchical RL and a heuristic algorithm. We find that MTSGI significantly outperforms all other baselines, and the learned prior model enables more efficient task inference compared to MSGI.

2 Preliminaries

Few-shot Reinforcement Learning A *task* is defined by an MDP $\mathcal{M}_G = (\mathcal{S}, \mathcal{A}, \mathcal{P}_G, \mathcal{R}_G)$ parameterized by a task parameter G with a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, transition dynamics $\mathcal{P}_G$, reward function $\mathcal{R}_G$. The goal of K-shot RL (Duan et al., 2016; Finn et al., 2017), is to efficiently solve a distribution of unseen test tasks $\mathcal{M}^{\text{test}}$ by learning and transferring the common knowledge from the training tasks $\mathcal{M}^{\text{train}}$. It is assumed that the training and test tasks do not overlap (*i.e.*, $\mathcal{M}^{\text{train}} \cap \mathcal{M}^{\text{test}} = \emptyset$) but share a certain commonality such that the knowledge learned from the training tasks may be helpful for learning the test tasks. For each task $\mathcal{M}_G$, the agent is given K

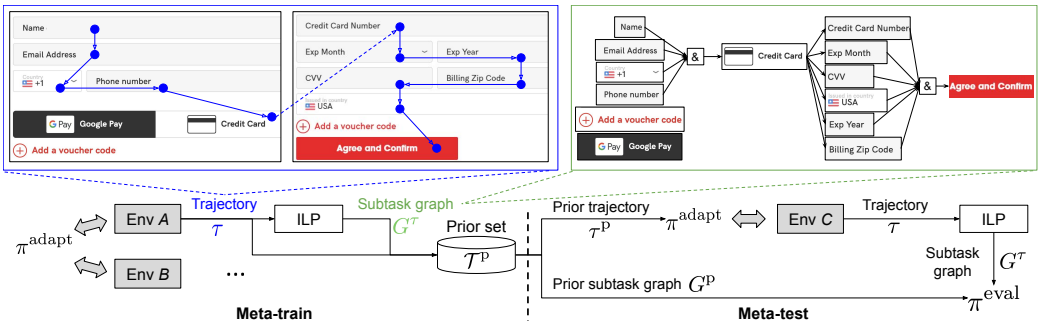

Figure 2: The overview of our algorithm and the example of agent's trajectory and the inferred subtask graph. In meta-train *(Left)*, the adaptation policy π^{adapt} interacts with the environment and collects the trajectory τ. The inductive logic programming (ILP) module takes as input the trajectory, and infers the task structure in terms of the subtask graph G^{τ}. The trajectory and the subtask graph are stored as a prior. In meta-testing *(Right)*, the adaptation policy incorporates the prior trajectory τ^{p} to efficiently explore the environment, and ILP module infers the subtask graph G^{τ} from the adaptation trajectory τ. Finally, the evaluation policy π^{eval} takes as input the prior and inferred subtask graphs (G^{p}, G^{τ}) to solve the test task.

steps budget for interacting with the environment. During meta-training, the goal of multi-task RL agent is to learn a prior (*i.e.*, slow-learning) over the training tasks $\mathcal{M}^{\text{train}}$. Then, the learned prior may be exploited during the meta-test to enable faster adaptation on unseen test tasks $\mathcal{M}^{\text{test}}$. For each task, the agent faces two phases: an *adaptation phase* where the agent learns a task-specific behavior (*i.e.*, fast-learning) for K environment steps, which often spans over multiple episodes, and a *evaluation phase* where the adapted behavior is evaluated. In the evaluation phase, the agent is not allowed to perform any form of learning, and agent's performance on the task $\mathcal{M}_G$ is measured in terms of the return:

$$\mathcal{R}_{\mathcal{M}_G}(\pi_{\phi_K}) = \mathbb{E}_{\pi_{\phi_K}, \mathcal{M}_G}\left[\sum_{t=1}^{H} r_t\right], \tag{1}$$

where π_{ϕ_K} is the policy after K update steps of adaptation, H is the horizon of evaluation phase, and r_t is the reward at time t in the evaluation phase.

3 Subtask Graph Inference Problem

The *subtask graph inference* problem (Sohn et al., 2019) is a few-shot RL problem where a task is parameterized by a set of subtasks and their dependencies. Formally, a task consists of N subtasks $\mathbf{\Phi} = \{\Phi^1, \ldots, \Phi^N\}$, and each subtask Φ^i is parameterized by a tuple $({\mathcal{S}_{\text{comp}}}^i, G^i_{\mathbf{c}}, G^i_{\mathbf{r}})$. The *goal state* ${\mathcal{S}_{\text{comp}}}^i \subset \mathcal{S}$ and *precondition* $G^i_{\mathbf{c}} : \mathcal{S} \to \{0, 1\}$ defines the condition that a subtask is *completed*: the current state should be contained in its goal states (*i.e.*, $\mathbf{s}_t \in \mathcal{S}^i_{\text{comp}}$) and the precondition should be satisfied (*i.e.*, $G^i_{\mathbf{c}}(\mathbf{s}_t) = 1$). If the precondition is not satisfied (*i.e.*, $G^i_{\mathbf{c}}(\mathbf{s}_t) = 0$), the subtask cannot be completed and the agent receives no reward even if the goal state is achieved. The *subtask reward function* $G^i_{\mathbf{r}}$ defines the amount of reward given to the agent when it *completes* the subtask i: $r_t \sim G^i_{\mathbf{r}}$. We note that the subtasks $\{\Phi^1, \ldots, \Phi^N\}$ are unknown to the agent. Thus, the agent should learn to infer the underlying task structure and complete the subtasks in an optimal order while satisfying the required preconditions.

State. In the subtask graph inference problem, it is assumed that the state input provides the high-level status of the subtasks. Specifically, the state consists of the followings: $\mathbf{s}_t = (\text{obs}_t, \mathbf{x}_t, \mathbf{e}_t, \text{step}_{\text{epi},t}, \text{step}_{\text{phase},t})$. The $\text{obs}_t \in \{0, 1\}^{W \times H \times C}$ is a visual observation of the environment. The completion vector $\mathbf{x}_t \in \{0, 1\}^N$ indicates whether each subtask is complete. The eligibility vector $\mathbf{e}_t \in \{0, 1\}^N$ indicates whether each subtask is eligible (*i.e.*, precondition is satisfied). Following the few-shot RL setting, the agent observes two scalar-valued time features: the remaining time steps until the episode termination $\text{step}_{\text{epi},t} \in \mathbb{R}$ and the remaining time steps until the phase termination $\text{step}_{\text{phase},t} \in \mathbb{R}$.

Options. For each subtask Φ^i, the agent can learn an option $\mathcal{O}^i$ (Sutton et al., 1999) that reaches the goal state of the subtask. Following Sohn et al. (2019), such options are pre-learned individually by maximizing the goal-reaching reward: $r_t = \mathbb{I}(\mathbf{s}_t \in \mathcal{S}^i_{\text{comp}})$. At time step t, we denote the option taken by the agent as $\mathbf{o}_t$ and the binary variable that indicates whether episode is terminated as d_t.

4 METHOD

We propose a novel Multi-Task Subtask Graph Inference (MTSGI) framework that can perform an efficient inference of latent task embedding (*i.e.*, subtask graph). The overall method is outlined in Figure 2. Specifically, in meta-training, MTSGI models the prior in terms of (1) adaptation trajectory τ and (2) subtask graph G from the agent's experience. In meta-testing, MTSGI samples (1) the prior trajectory τ^{p} for more efficient exploration in adaptation and (2) the prior subtask graph G^{p} for more accurate task inference.

4.1 MULTI-TASK ADAPTATION POLICY

The goal of *adaptation policy* is to efficiently explore and gather the information about the task. Intuitively, if the adaptation policy completes more diverse subtasks, then it can provide more data to the task inference module (ILP), which in turn can more accurately infer the task structure. To this end, we extend the upper confidence bound (UCB)-based adaptation policy proposed in Sohn et al. (2019) as follows:

$$\pi^{\text{adapt}}(o = \mathcal{O}^i \mid s) \propto \exp\left(r^i + \frac{\log\left(\sum_j n^j\right)}{\sqrt{2}n^i}\right), \tag{2}$$

where r^i is the empirical mean of the reward received after executing subtask i and n^i is the number of times subtask i has been executed within the current task. Note that the exploration parameters $\{r^i, n^i\}_{i=1}^N$ can be computed from the agent's trajectory. In meta-train, the exploration parameters are initialized to zero when a new task is sampled. In meta-test, the exploration parameters are initialized with those of the sampled prior. Intuitively, this helps the agent visit novel states that were unseen during meta-training.

Algorithm 1 Meta-training: learning the prior

Require: Adaptation policy π^{adapt}
Ensure: Prior set $\mathcal{T}^{\text{p}}$
1: $\mathcal{T}^{\text{p}} \leftarrow \emptyset$
2: **for** each task $\mathcal{M} \in \mathcal{M}^{\text{train}}$ **do**
3: Rollout adaptation policy: $\tau = \{\mathbf{s}_t, \mathbf{o}_t, r_t, d_t\}_{t=1}^K \sim \pi^{\text{adapt}}$ in $\mathcal{M}$
4: $G^\tau = \arg\max_G p(\tau|G)$
5: $\pi^{\text{eval}} = \text{GRProp}(G^\tau)$
6: Evaluate the agent: $\tau^{\text{eval}} \sim \pi^{\text{eval}}$ in $\mathcal{M}$
7: Update prior: $\mathcal{T}^{\text{p}} \leftarrow \mathcal{T}^{\text{p}} \cup (G^\tau, \tau)$
8: **end for**

Algorithm 2 meta-testing: multi-task SGI

Require: Adaptation policy π^{adapt}, prior set $\mathcal{T}^{\text{p}}$
1: **for** each task $\mathcal{M} \in \mathcal{M}^{\text{test}}$ **do**
2: Sample prior: $(G^{\text{p}}, \tau^{\text{p}}) \sim p(\mathcal{T}^{\text{p}})$
3: Rollout adaptation policy: $\tau = \{\mathbf{s}_t, \mathbf{o}_t, r_t, d_t\}_{t=1}^K \sim \pi^{\text{adapt}}$ in $\mathcal{M}$
4: $G^\tau = \arg\max_G p(\tau|G)$
5: $\pi^{\text{eval}}(\cdot|\tau, \tau^{\text{p}}) \propto \text{GRProp}(\cdot|G^\tau)^\alpha \text{GRProp}(\cdot|G^{\text{p}})^{(1-\alpha)}$
6: Evaluate the agent: $\tau^{\text{eval}} \sim \pi^{\text{eval}}$ in task $\mathcal{M}$
7: **end for**

4.2 META-TRAIN: LEARNING THE PRIOR SUBTASK GRAPH

Let τ be an adaptation trajectory of the agent for K steps. The goal is to infer the latent subtask graph G for the given training task $\mathcal{M}_G \in \mathcal{M}^{\text{train}}$, specified by preconditions $G_{\mathbf{c}}$ and subtask rewards $G_{\mathbf{r}}$. We find the maximum-likelihood estimate (MLE) of $G = (G_{\mathbf{c}}, G_{\mathbf{r}})$ that maximizes the likelihood of the adaptation trajectory τ:

$$\widehat{G}^{\text{MLE}} = \arg\max_{G_{\mathbf{c}}, G_{\mathbf{r}}} p(\tau|G_{\mathbf{c}}, G_{\mathbf{r}}). \tag{3}$$

Following Sohn et al. (2019), we infer the precondition $G_{\mathbf{c}}$ and the subtask reward $G_{\mathbf{r}}$ as follows (See Appendix for the detailed derivation):

$$\widehat{G}_{\mathbf{c}}^{\text{MLE}} = \arg\max_{G_{\mathbf{c}}} \prod_{t=1}^{H} p(\mathbf{e}_t|\mathbf{x}_t, G_{\mathbf{c}}), \quad \widehat{G}_{\mathbf{r}}^{\text{MLE}} = \arg\max_{G_{\mathbf{r}}} \prod_{t=1}^{H} p(r_t|\mathbf{e}_t, \mathbf{o}_t, G_{\mathbf{r}}). \tag{4}$$

where $\mathbf{e}_t$ is the eligibility vector, $\mathbf{x}_t$ is the completion vector, $\mathbf{o}_t$ is the option taken, r_t is the reward at time step t.

4.2.1 PRECONDITION INFERENCE

The problem in Equation (4) is known as the inductive logic programming (ILP) problem that finds a boolean function that satisfies all the indicator functions. Specifically, $\{\mathbf{x}_t\}_{t=1}^H$ forms binary vector inputs to programs, and $\{e_t^i\}_{t=1}^H$ forms Boolean-valued outputs of the i-th program that predicts the eligibility of the i-th subtask. We use the *classification and regression tree* (CART) to infer the precondition function $f_{G_\mathbf{e}} : \mathbf{x} \rightarrow \mathbf{e}$ for each subtask based on Gini impurity (Breiman, 1984). Intuitively, the constructed decision tree is the simplest boolean function approximation for the given input-output pairs $\{\mathbf{x}_t, \mathbf{e}_t\}$. The decision tree is converted to a logic expression (*i.e.*, precondition) in sum-of-product (SOP) form to build the subtask graph.

4.2.2 SUBTASK REWARD INFERENCE

To infer the subtask reward $\widehat{G}_\mathbf{r}^{\text{MLE}}$ in Equation (4), we model the reward for i-th subtask as a Gaussian distribution: $G_\mathbf{r}^i \sim \mathcal{N}(\widehat{\mu}^i, \widehat{\sigma}^i)$. Then, the MLE of subtask reward is given as the empirical mean and variance of the rewards received after taking the eligible option $\mathcal{O}^i$ in adaptation phase:

$$\widehat{\mu}_{\text{MLE}}^i = \mathbb{E}\left[r_t | \mathbf{o}_t = \mathcal{O}^i, \mathbf{e}_t^i = 1\right], \quad \widehat{\sigma^2}_{\text{MLE}}^i = \mathbb{E}\left[(r_t - \widehat{\mu}_{\text{MLE}}^i)^2 | \mathbf{o}_t = \mathcal{O}^i, \mathbf{e}_t^i = 1\right], \tag{5}$$

where $\mathcal{O}^i$ is the option corresponding to the i-th subtask. Algorithm 1 outlines the meta-training process.

4.3 EVALUATION: GRAPH-REWARD PROPAGATION POLICY

In both meta-training and meta-testing, the agent's adapted behavior is evaluated during the test phase. Following Sohn et al. (2019), we adopted the graph reward propagation (GRProp) policy as an evaluation policy π^{eval} that takes as input the inferred subtask graph $\widehat{G}$ and outputs the subtasks to execute to maximize the return. Intuitively, the GRProp policy approximates a subtask graph to a differentiable form such that we can compute the gradient of return with respect to the completion vector to measure how much each subtask is likely to increase the return. Due to space limitations, we give a detail of the GRProp policy in Appendix. The overall meta-training process is summarized in Appendix.

4.4 META-TESTING: MULTI-TASK TASK INFERENCE

4.4.1 PRIOR SAMPLING

In meta-testing, MTSGI first chooses the prior task that most resembles the given evaluation task. Specifically, we define the pair-wise similarity between a prior task $\mathcal{M}_G^{\text{prior}}$ and the evaluation task $\mathcal{M}_G$ as follows:

$$\text{sim}\left(\mathcal{M}_G, \mathcal{M}_G^{\text{prior}}\right) = F_\beta\left(\mathbf{\Phi}, \mathbf{\Phi}^{\text{prior}}\right) + \kappa R\left(\tau^{\text{prior}}\right), \tag{6}$$

where F_β is the F-score with weight parameter β, $\mathbf{\Phi}$ is the subtask set of $\mathcal{M}_G$, $\mathbf{\Phi}^{\text{prior}}$ is the subtask set of $\mathcal{M}_G^{\text{prior}}$, $R\left(\tau^{\text{prior}}\right)$ is the agent's empirical performance on the prior task $\mathcal{M}_G^{\text{prior}}$, and κ is a scalar-valued weight which we used $\kappa = 1.0$ in experiment. F_β measures how many subtasks overlap between current and prior tasks in terms of precision and recall as follows:

$$F_\beta = \left(1 + \beta^2\right) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}, \tag{7}$$

$$\text{Precision} = |\mathbf{\Phi} \cap \mathbf{\Phi}^{\text{prior}}| / |\mathbf{\Phi}^{\text{prior}}|, \quad \text{Recall} = |\mathbf{\Phi} \cap \mathbf{\Phi}^{\text{prior}}| / |\mathbf{\Phi}|. \tag{8}$$

We used $\beta = 10$ to assign a higher weight to the current task (*i.e.*, recall) than the prior task (*i.e.*, precision).

4.4.2 MULTI-TASK SUBTASK GRAPH INFERENCE

Let τ be the adaptation trajectory, and τ^{p} be the sampled prior adaptation trajectory. Then, we model our evaluation policy as follows:

$$\pi(o|s, \tau, \tau^{\text{p}}) \simeq \pi(o|s, G^\tau)^\alpha \pi(o|s, G^{\text{p}})^{(1-\alpha)}. \tag{9}$$

Due to the limited space, we include the detailed derivation of Equation (9) in Appendix. Finally, we deploy the GRProp policy as a contextual policy:

$$\pi^{\text{eval}}(\cdot|\tau, \tau^{\text{p}}) = \text{GRProp}(\cdot|G^{\tau})^{\alpha}\text{GRProp}(\cdot|G^{\text{p}})^{(1-\alpha)}. \tag{10}$$

Note that Equation (10) is the weighted sum of the logits of two GRProp policies induced by prior τ^{p} and current experience τ. We claim that such form of ensemble induces the positive transfer in compositional tasks. Intuitively, ensembling GRProp is taking a union of preconditions since GRProp assigns a positive logit to task-relevant subtask and non-positive logit to other subtasks. As motivated in the Introduction, related tasks often share the task-relevant preconditions; thus, taking the union of task-relevant preconditions is likely to be a positive transfer and improve the generalization. The pseudo-code of the multi-task subtask graph inference process is summarized in Algorithm 2.

5 Related Work

Web navigating RL agent Previous work introduced MiniWoB (Shi et al., 2017) and MiniWoB++ (Liu et al., 2018) benchmarks that are manually curated sets of simulated toy environments for the web navigation problem. They formulated the problem as acting on a page represented as a Document Object Model (DOM), a hierarchy of objects in the page. The agent is trained with human demonstrations and online episodes in an RL loop. Jia et al. (2019) proposed a graph neural network based DOM encoder and a multi-task formulation of the problem similar to this work. Gur et al. (2018) introduced a manually-designed curriculum learning method and an LSTM based DOM encoder. DOM level representations of web pages pose a significant sim-to-real gap as simulated websites are considerably smaller (100s of nodes) compared to noisy real websites (1000s of nodes). As a result, these models are trained and evaluated on the same simulated environments which are difficult to deploy on real websites. Our work formulates the problem as abstract web navigation on real websites where the objective is to learn a latent subtask dependency graph similar to a sitemap of websites. We propose a multi-task training objective that generalizes from a fixed set of real websites to unseen websites without any demonstration, illustrating an agent capable of navigating real websites for the first time.

Meta-reinforcement learning To tackle the few-shot RL problem, researchers have proposed two broad categories of meta-RL approaches: RNN- and gradient-based methods. The RNN-based meta-RL methods (Duan et al., 2016; Wang et al., 2016; Hochreiter et al., 2001) encode the common knowledge of the task into the hidden states and the parameters of the RNN. The gradient-based meta-RL methods (Finn et al., 2017; Nichol et al., 2018; Gupta et al., 2018; Finn et al., 2018; Kim et al., 2018) encode the task embedding in terms of the initial policy parameter for fast adaptation through meta gradient. Existing meta-RL approaches, however, often require a large amount of environment interaction due to the long-horizon nature of the few-shot RL tasks. Our work instead explicitly infers the underlying task parameter in terms of subtask graph, which can be efficiently inferred using the inductive logic programming (ILP) method and be transferred across different, unseen tasks.

More Related Works Please refer to the Appendix for further discussions about other related works.

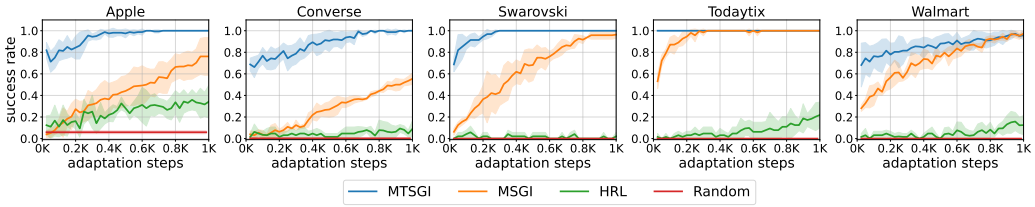

Figure 3: The success rate of the compared methods in the test phase in terms of the environment step during the adaptation phase on *SymWoB* domain. See Appendix for the results on other tasks.

6 EXPERIMENT

6.1 DOMAINS

6.1.1 MINING

Mining (Sohn et al., 2018) is a 2D grid-world domain inspired by Minecraft game where the agent receives a reward by picking up raw materials in the world or crafting items with raw materials. The subtask dependency is defined according to the crafting recipe implemented in the game. Following Sohn et al. (2018), we used the pre-generated train/test task splits generated with four different random seeds. Each split set consists of 3200 training tasks and 440 testing tasks for meta-training and meta-testing, respectively. We report the performance averaged over the four task split sets.

6.1.2 SYMWOB

We implement a symbolic version of the checkout process on the 20 real-world websites such as **Amazon**, **Converse**, and **Walmart**, etc.

Subtask and option policy. Each actionable web element (*e.g.*, text field, button drop-down list, and hyperlink) is considered as a subtask. We assume the agent has pre-learned the option policies that correctly interact with each element (*e.g.*, click the button or fill out the text field). Thus, the agent should learn a policy over the option.

Completion and eligibility. For each subtask, the completion and eligibility are determined based on the status of the corresponding web element. For example, the subtask of a text field is *completed* if the text field is filled with the correct information, and the subtask of a `confirm_credit_info` button is *eligible* if all the required subtasks (*i.e.*, filling out credit card information) are completed. Executing an option will complete the corresponding subtask *only if* the subtask is eligible.

Reward function and episode termination. The agent may receive a non-zero reward only at the end of the episode (*i.e.*, sparse-reward task). When the episode terminates due to the time budget, the agent may not receive any reward. Otherwise, the following two types of subtasks terminate the episode and give a non-zero reward upon completion:

- **Goal subtask** refers to the button that completes the order (See the green boxes in Figure 1). Completing this subtask gives the agent a +5 reward, and the episode is terminated.
- **Distractor subtask** does not contribute to solving the given task but terminates the episode with a -1 reward. It models the web elements that lead to external web pages such as `Contact_Us` button in Figure 1.

Transition dynamics. The transition dynamics follow the dynamics of the actual website. Each website consists of multiple web pages. The agent may only execute the subtasks that are currently visible (*i.e.*, on the current web page) and can navigate to the next web page only after filling out all the required fields and clicking the continue button. The goal subtask is present in the last web page; thus, the agent must learn to navigate through the web pages to solve the task.

For more details about each task, please refer to Appendix.

6.2 AGENTS

We compared the following algorithms in the experiment.

- MTSGI (Ours): our multi-task SGI agent
- MSGI (Sohn et al., 2019): SGI agent without multi-task learning
- HRL: an Option (Sutton et al., 1999)-based proximal policy optimization (PPO) (Schulman et al., 2017) agent with the gated rectifier unit (GRU)
- Random: a heuristic policy that uniform randomly executes an eligible subtask

More details on the architectures and the hyperparameters can be found in Appendix.

Meta-training In *SymWoB*, for each task chosen for a meta-testing, we randomly sampled N_{train} tasks among the remaining 19 tasks and used it for meta-training. We used $N_{\text{train}} = 1$ in the experiment (See Figure 8 for the impact of the choice of N_{train}). For example, we meta-trained our MTSGI on **Amazon** and meta-tested on **Samsung**. For *Mining*, we used the train/test task split

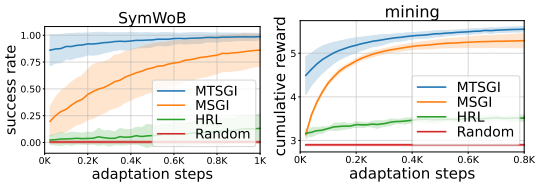

Figure 4: The performance of the compared methods in terms of the adaptation steps averaged over all the tasks in *SymWoB (Left)*and *Mining (Right)*domains.

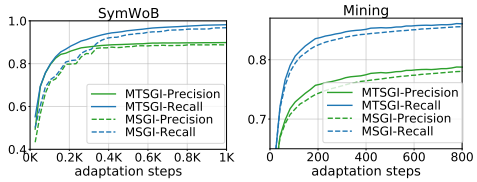

Figure 5: The precision and recall of the subtask graphs inferred by MTSGI and MSGI on *SymWoB* and *Mining*.

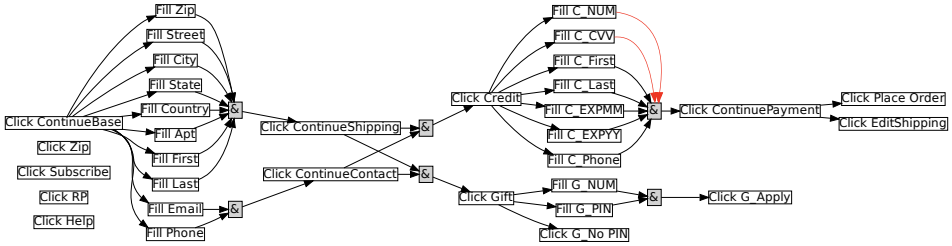

Figure 6: The visualization of the subtask graph inferred by our MTSGI after 1,000 steps of environment interaction on **Walmart** domain. Compared to the ground-truth subtask graph (not available to the agent), there was no error in the nodes and only two missing edges (in red). See Appendix for the progression of the inferred subtask graph with varying adaptation steps.

provided in the environment. The RL agents (*e.g.*, HRL) were individually trained on each test task; the policy was initialized when a new task is sampled and trained during the adaptation phase. All the experiments were repeated with four random seeds, where different training tasks were sampled for different seeds.

6.3 Result: Few-shot Generalization Performance

Figure 3 and Figure 4 show the few-shot generalization performance of the compared methods on *SymWoB* and *Mining*. In Figure 3, MTSGI achieves more than 80% zero-shot success rate (*i.e.*, success rate at x-axis=0) on four out of five tasks, which is significantly higher than the zero-shot performance of MSGI. This indicates that the prior learned from the training task significantly improves the subtask graph inference and in turn improves the multi-task evaluation policy. Moreover, our MTSGI can learn a near-optimal policy on all the tasks after only 1,000 steps of environment interactions, demonstrating that the proposed multi-task learning scheme enables fast adaptation. Even though the MSGI agent is learning each task from scratch, it still outperforms the HRL and Random agents. This shows that explicitly inferring the underlying task structure and executing the predicted subtask graph is significantly more effective than learning the policy from the reward signal (*i.e.*, HRL) on complex compositional tasks. Given the pre-learned options, HRL agent can slightly improve the success rate during the adaptation via PPO update. However, training the policy from the sparse reward requires a large number of interactions especially for the tasks with many distractors (*e.g.*, **Converse** and **Swarovski**).

6.4 Analysis on the Inferred Subtask Graph

We compare the inferred subtask graph with the ground-truth subtask graph. Figure 6 shows the subtask graph inferred by MTSGI in **Walmart**. We can see that MTSGI can accurately infer the subtask graph; the inferred subtask graph is missing only two preconditions (shown in red color) of `Click_Continue_Payment` subtask.We note that such a small error in the subtask graph has a negligible effect as shown in Figure 3: *i.e.*, MTSGI achieves near-optimal performance on **Walmart** after 1,000 steps of adaptation. Figure 5 measures the precision and recall of the inferred precondition (*i.e.*, the edge of the graph). First, both MTSGI and MSGI achieve high precision and recall after only a few hundred of adaptation. Also, MTSGI outperforms MSGI in the early stage of adaptation. This clearly demonstrates that the MTSGI can perform more accurate task inference due to the prior learned from the training tasks.

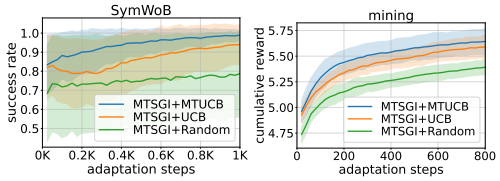

Figure 7: Comparison of different exploration strategies for MTSGI used in adaptation phase for *SymWoB* and *Mining*.

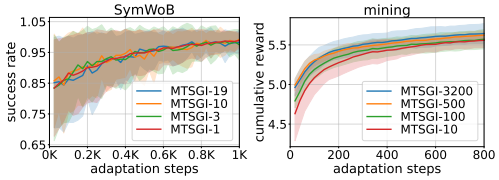

Figure 8: Comparison of different number of priors for MTSGI on *SymWoB* and *Mining*.

6.5 Ablation Study: Effect of Exploration Strategy

In this section, we investigate the effect of various exploration strategies on the performance of MTSGI. We compared the following three adaptation policies:

- Random: A policy that uniformly randomly executes any eligible subtask.
- UCB: The UCB policy defined in Equation (2) that aims to execute the novel subtask. The exploration parameters are initialized to zero when a new task is sampled.
- MTUCB (Ours): Our multi-task extension of UCB policy. When a new task is sampled, the exploration parameter is initialized with those of the sampled prior.

Figure 7 summarizes the result on *SymWoB* and *Mining* domain, respectively. Using the more sophisticated exploration policy such as MTSGI+UCB or MTSGI+MTUCB improved the performance of MTSGI compared to MTSGI+Random, which was also observed in Sohn et al. (2019). This is because better exploration helps the adaptation policy collect more data for logic induction by executing more diverse subtasks. In turn, this results in more accurate subtask graph inference and better performance. Also, MTSGI+MTUCB outperforms MTSGI+UCB on both domains. This indicates that transferring the exploration parameters makes the agent's exploration more efficient in meta-testing. Intuitively, the transferred exploration counts inform the agent which subtasks were *under-explored* during meta-training, such that the agent can focus more on exploring those in meta-testing.

6.6 Ablation Study: Effect of the prior set size

MTSGI learns the prior from the training tasks. We investigated how many training tasks are required for MTSGI to learn a good prior for transfer learning. Figure 8 compares the performance of MTSGI with the varying number of training tasks: 1, 3, 10, 19 tasks for *SymWoB* and 10, 100, 500, 3200 tasks for *Mining*. The training tasks are randomly subsampled from the entire training set. The result shows that training on a larger number of tasks generally improves the performance, but it saturates (*e.g.*, $|\mathcal{M}^{\text{train}}| = 1$ for *SymWoB* and $|\mathcal{M}^{\text{train}}| = 500$ for *Mining*). *Mining* generally requires more number of training tasks than *SymWoB* because the agent is required to solve 440 different tasks in *Mining* while *SymWoB* was evaluated on 20 tasks; the agent is required to capture a wider range of task distribution in *Mining* than *SymWoB*. Also, we note that MTSGI can still adapt much more efficiently than all other baseline methods even when only a small number of training tasks are available (*e.g.*, one task for *SymWoB* and ten tasks for *Mining*).

7 Conclusion

We introduce a multi-task RL extension of the subtask graph inference framework that can quickly adapt to the unseen tasks by modeling the prior of subtask graph from the training tasks and transferring it to the test tasks. The empirical results demonstrate that our MTSGI achieves strong zero-shot and few-shot generalization performance on 2D grid-world and complex web navigation domains by transferring the common knowledge learned in the training tasks to the unseen ones in terms of subtask graph.

In this work, we have assumed that the subtasks and the corresponding options are pre-learned and that the environment provides a high-level status of each subtask (*e.g.*, whether the web element is filled in with the correct information). In future work, our approach may be extended to a more general setting where the relevant subtask structure is fully learned from (visual) observations, and the corresponding options are autonomously discovered.

REFERENCES

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *ICML*, 2017.

Leo Breiman. *Classification and regression trees*. Routledge, 1984.

Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. Gated-attention architectures for task-oriented language grounding. In *AAAI*, 2018.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International Conference on Machine Learning*, pp. 794–803. PMLR, 2018.

Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl ^{2}: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1126–1135. JMLR. org, 2017.

Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. In *NeurIPS*, pp. 9516–9527, 2018.

Behzad Ghazanfari and Matthew E Taylor. Autonomous extracting a hierarchical structure of tasks in reinforcement learning and multi-task reinforcement learning. *arXiv preprint arXiv:1709.04579*, 2017.

Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. *arXiv preprint arXiv:1802.07245*, 2018.

Izzeddin Gur, Ulrich Rueckert, Aleksandra Faust, and Dilek Hakkani-Tur. Learning to navigate the web. *arXiv preprint arXiv:1812.09195*, 2018.

Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pp. 87–94. Springer, 2001.

Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020. URL `https://arxiv.org/abs/2006.00979`.

De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Li Fei-Fei, Silvio Savarese, and Juan Carlos Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. *arXiv preprint arXiv:1807.03480*, 2018.

Sheng Jia, Jamie Ryan Kiros, and Jimmy Ba. DOM-q-NET: Grounded RL on structured language. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HJgd1nAqFX`.

Taesup Kim, Jaesik Yoon, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian model-agnostic meta-learning. *arXiv preprint arXiv:1806.03836*, 2018.

George Konidaris and Andrew Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pp. 489–496, 2006.

Alessandro Lazaric. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*, pp. 143–173. Springer, 2012.

Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Transfer of samples in batch reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 544–551, 2008.

Xingyu Lin, Harjatin Singh Baweja, George Kantor, and David Held. Adaptive auxiliary task weighting for reinforcement learning. *Advances in neural information processing systems*, 32, 2019.

Changsong Liu, Shaohua Yang, Sari Iaba-Sadiya, Nishant Shukla, Yunzhong He, Song-chun Zhu, and Joyce Chai. Jointly learning grounded task structures from language instruction and visual demonstration. In *EMNLP*, 2016.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. *arXiv preprint arXiv:1802.08802*, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Stephen Muggleton. Inductive logic programming. *New Gen. Comput.*, 8(4):295–318, February 1991. ISSN 0288-3635. doi: 10.1007/BF03037089. URL `http://dx.doi.org/10.1007/BF03037089`.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *ICML*, 2017.

Lerrel Pinto and Abhinav Gupta. Learning to push by grasping: Using multiple tasks for effective learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 2161–2168. IEEE, 2017.

Earl D Sacerdoti. The nonlinear nature of plans. Technical report, STANFORD RESEARCH INST MENLO PARK CA, 1975a.

Earl D Sacerdoti. A structure for plans and behavior. Technical report, SRI INTERNATIONAL MENLO PARK CA ARTIFICIAL INTELLIGENCE CENTER, 1975b.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, pp. 3135–3144, 2017.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Sungryull Sohn, Junhyuk Oh, and Honglak Lee. Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies. In *NeurIPS*, pp. 7156–7166, 2018.

Sungryull Sohn, Hyunjae Woo, Jongwook Choi, and Honglak Lee. Meta reinforcement learning with autonomous inference of subtask dependencies. In *International Conference on Learning Representations*, 2019.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Austin Tate. Generating project networks. In *Proceedings of the 5th international joint conference on Artificial intelligence-Volume 2*, pp. 888–893. Morgan Kaufmann Publishers Inc., 1977.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th international conference on Machine learning*, pp. 1015–1022, 2007.

Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Neural task programming: Learning to generalize across hierarchical tasks. *arXiv preprint arXiv:1710.01813*, 2017.

Haonan Yu, Haichao Zhang, and Wei Xu. A deep compositional framework for human-like language acquisition in virtual environment. *arXiv preprint arXiv:1703.09831*, 2017.

Yu Zhang and Dit-Yan Yeung. A regularization approach to learning task relationships in multitask learning. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(3):1–31, 2014.

Appendix: Fast Inference and Transfer of Compositional Task Structures for Few-shot Task Generalization

A DERIVATION OF SUBTASK GRAPH INFERENCE

Following Sohn et al. (2019), we formulate the problem of inferring the precondition $G_{\mathbf{c}}$ and the subtask reward $G_{\mathbf{r}}$ as a maximum likelihood estimation (MLE) problem. Let $\tau_K = \{\mathbf{s}_1, \mathbf{o}_1, r_1, d_1, \ldots, \mathbf{s}_K\}$ be an adaptation trajectory of the adaptation policy $\pi_\theta^{\text{adapt}}$ for K steps in adaptation phase. The goal is to infer the subtask graph G for this task, specified by preconditions $G_{\mathbf{c}}$ and subtask rewards $G_{\mathbf{r}}$. Consider a subtask graph G with subtask reward $G_{\mathbf{r}}$ and precondition $G_{\mathbf{c}}$. The maximum-likelihood estimate (MLE) of latent variables G, conditioned on the trajectory τ_H can be written as

$$\widehat{G}^{\text{MLE}} = \arg\max_{G_{\mathbf{c}}, G_{\mathbf{r}}} p(\tau_K | G_{\mathbf{c}}, G_{\mathbf{r}}). \tag{11}$$

The likelihood term can be expanded as

$$p(\tau_K | G_{\mathbf{c}}, G_{\mathbf{r}}) = p(\mathbf{s}_1 | G_{\mathbf{c}}) \prod_{t=1}^{K} \pi_\theta\left(\mathbf{o}_t | \tau_t\right) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{o}_t, G_{\mathbf{c}}) p(r_t | \mathbf{s}_t, \mathbf{o}_t, G_{\mathbf{r}}) p(d_t | \mathbf{s}_t, \mathbf{o}_t) \tag{12}$$

$$\propto p(\mathbf{s}_1 | G_{\mathbf{c}}) \prod_{t=1}^{K} p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{o}_t, G_{\mathbf{c}}) p(r_t | \mathbf{s}_t, \mathbf{o}_t, G_{\mathbf{r}}), \tag{13}$$

where we dropped the terms that are independent of G. From the definitions in **Subtask Graph Inference Problem** section, precondition $G_{\mathbf{c}}$ defines the mapping $\mathbf{x} \mapsto \mathbf{e}$, and the subtask reward $G_{\mathbf{r}}$ determines the reward as $r_t \sim {G_{\mathbf{r}}}^i$ if subtask i is eligible (*i.e.*, $\mathbf{e}_t^i = 1$) and option $\mathcal{O}^i$ is executed at time t. Therefore, we have

$$\widehat{G}^{\text{MLE}} = (\widehat{G}_{\mathbf{c}}^{\text{MLE}}, \widehat{G}_{\mathbf{r}}^{\text{MLE}}) = \left(\arg\max_{G_{\mathbf{c}}} \prod_{t=1}^{K} p(\mathbf{e}_t | \mathbf{x}_t, G_{\mathbf{c}}),\ \arg\max_{G_{\mathbf{r}}} \prod_{t=1}^{K} p(r_t | \mathbf{e}_t, \mathbf{o}_t, G_{\mathbf{r}}) \right). \tag{14}$$

Below we explain how to compute the estimate of preconditions $\widehat{G}_{\mathbf{c}}^{\text{MLE}}$ and subtask rewards $\widehat{G}_{\mathbf{r}}^{\text{MLE}}$.

Precondition inference via logic induction From the definition in **Subtask Graph Inference Problem** section, the mapping from a precondition $G_{\mathbf{c}}$ and a completion vector $\mathbf{x}$ to an eligibility vector $\mathbf{e} = f_{G_{\mathbf{c}}}(\mathbf{x})$ is a deterministic function (*i.e.*, precondition function). Therefore, we can rewrite $\widehat{G}_{\mathbf{c}}^{\text{MLE}}$ in Eq.(14) as:

$$\widehat{G}_{\mathbf{c}}^{\text{MLE}} = \arg\max_{G_{\mathbf{c}}} \prod_{t=1}^{K} \mathbb{I}(\mathbf{e}_t = f_{G_{\mathbf{c}}}(\mathbf{x}_t)), \tag{15}$$

where $\mathbb{I}(\cdot)$ is the indicator function. Since the eligibility $\mathbf{e}$ is factored, the precondition function $f_{{G_{\mathbf{c}}}^i}$ for each subtask is inferred independently. This problem of finding a boolean function that satisfies all the indicator functions in Eq.(15) (*i.e.*, $\prod_{t=1}^{K} \mathbb{I}(\mathbf{e}_t = f_{G_{\mathbf{c}}}(\mathbf{x}_t)) = 1$) is formulated as an *inductive logic programming* (ILP) problem (Muggleton, 1991). Specifically, $\{\mathbf{x}_t\}_{t=1}^K$ forms binary vector inputs to programs, and $\{e_t^i\}_{t=1}^K$ forms Boolean-valued outputs of the i-th program that denotes the eligibility of the i-th subtask. We use the *classification and regression tree* (CART) to infer the precondition function $f_{G_{\mathbf{c}}}$ for each subtask based on Gini impurity (Breiman, 1984). Intuitively, the constructed decision tree is the simplest boolean function approximation for the given input-output pairs $\{\mathbf{x}_t, \mathbf{e}_t\}$. Then, we convert it to a logic expression (*i.e.*, precondition) in sum-of-product (SOP) form to build the subtask graph.

Subtask reward inference To infer the subtask reward function $\widehat{G}_{\mathbf{r}}^{\text{MLE}}$ in Eq.(14), we model each component of subtask reward as a Gaussian distribution $G_{\mathbf{r}}^i \sim \mathcal{N}(\widehat{\mu}^i, \widehat{\sigma}^i)$. Then, $\widehat{\mu}_{\text{MLE}}^i$ becomes the empirical mean of the rewards received after taking the eligible option $\mathcal{O}^i$ in the trajectory τ_K:

$$\widehat{G}_{\mathbf{r}}^{\text{MLE},i} = \widehat{\mu}_{\text{MLE}}^i = \mathbb{E}\left[r_t | \mathbf{o}_t = \mathcal{O}^i, \mathbf{e}_t^i = 1\right] = \frac{\sum_{t=1}^{K} r_t \mathbb{I}(\mathbf{o}_t = \mathcal{O}^i, \mathbf{e}_t^i = 1)}{\sum_{t=1}^{K} \mathbb{I}(\mathbf{o}_t = \mathcal{O}^i, \mathbf{e}_t^i = 1)}. \tag{16}$$

B DETAILS OF GRPROP POLICY

For self-containedness, we repeat the description of GRProp policy in Sohn et al. (2019).

Intuitively, GRProp policy modifies the subtask graph to a differentiable form such that we can compute the gradient of modified return with respect to the subtask completion vector in order to measure how much each subtask is likely to increase the modified return. Let $\mathbf{x}_t$ be a completion vector and $G_{\mathbf{r}}$ be a subtask reward vector (see **Subtask Graph Inference Problem** section for definitions). Then, the sum of reward until time-step t is given as:

$$U_t = {G_{\mathbf{r}}}^\top \mathbf{x}_t. \tag{17}$$

We first modify the reward formulation such that it gives a half of subtask reward for satisfying the preconditions and the rest for executing the subtask to encourage the agent to satisfy the precondition of a subtask with a large reward:

$$\widehat{U}_t = {G_{\mathbf{r}}}^\top (\mathbf{x}_t + \mathbf{e}_t)/2. \tag{18}$$

Let y^j_{AND} be the output of j-th AND node. The eligibility vector $\mathbf{e}_t$ can be computed from the subtask graph G and $\mathbf{x}_t$ as follows:

$$e^i_t = \underset{j \in Child_i}{\text{OR}} \left(y^j_{AND} \right), \quad y^j_{AND} = \underset{k \in Child_j}{\text{AND}} \left(\widehat{x}^{j,k}_t \right), \quad \widehat{x}^{j,k}_t = x^k_t w^{j,k} + \text{NOT}(x^k_t)(1 - w^{j,k}), \tag{19}$$

where $w^{j,k} = 0$ if there is a NOT connection between j-th node and k-th node, otherwise $w^{j,k} = 1$. Intuitively, $\widehat{x}^{j,k}_t = 1$ when k-th node does not violate the precondition of j-th node. The logical AND, OR, and NOT operations in Equation (19) are substituted by the smoothed counterparts as follows:

$$p^i = \lambda_{\text{or}} \widetilde{e}^i + (1 - \lambda_{\text{or}})\, x^i, \tag{20}$$

$$\widetilde{e}^i = \underset{j \in Child_i}{\widetilde{\text{OR}}} \left(\widetilde{y}^j_{AND} \right), \tag{21}$$

$$\widetilde{y}^j_{AND} = \underset{k \in Child_j}{\widetilde{\text{AND}}} \left(\hat{x}^{j,k} \right), \tag{22}$$

$$\hat{x}^{j,k} = w^{j,k} p^k + (1 - w^{j,k}) \widetilde{\text{NOT}} \left(p^k \right), \tag{23}$$

where $\mathbf{x} \in \mathbb{R}^d$ is the input completion vector,

$$\widetilde{\text{OR}}\,(\mathbf{x}) = \text{softmax}(w_{\text{or}} \mathbf{x}) \cdot \mathbf{x}, \tag{24}$$

$$\widetilde{\text{AND}}\,(\mathbf{x}) = \frac{\zeta(\mathbf{x}, w_{\text{and}})}{\zeta(||\mathbf{x}||, w_{\text{and}})}, \tag{25}$$

$$\widetilde{\text{NOT}}\,(\mathbf{x}) = -w_{\text{not}} \mathbf{x}, \tag{26}$$

$||\mathbf{x}|| = d$, $\zeta(\mathbf{x}, \beta) = \frac{1}{\beta} \log(1 + \exp(\beta \mathbf{x}))$ is a soft-plus function, and $\lambda_{\text{or}} = 0.6, w_{\text{or}} = 2, w_{\text{and}} = 3, w_{\text{not}} = 2$ are the hyper-parameters of GRProp. Note that we slightly modified the implementation of $\widetilde{\text{OR}}$ and $\widetilde{\text{AND}}$ from sigmoid and hyper-tangent functions in (Sohn et al., 2018) to softmax and softplus functions for better performance. With the smoothed operations, the sum of smoothed and modified reward is given as:

$$\widetilde{U}_t = {G_{\mathbf{r}}}^\top \mathbf{p}, \tag{27}$$

where $\mathbf{p} = [p^1, \ldots, p^d]$ and p^i is computed from Equation (20). Finally, the graph reward propagation policy is a softmax policy,

$$\pi(\mathbf{o}_t | G, \mathbf{x}_t) = \text{Softmax} \left(T \nabla_{\mathbf{x}_t} \widetilde{U}_t \right) = \text{Softmax} \left(T {G_{\mathbf{r}}}^\top (\lambda_{\text{or}} \nabla_{\mathbf{x}_t} \widetilde{\mathbf{e}}_t + (1 - \lambda_{\text{or}})) \right), \tag{28}$$

where we used the softmax temperature $T = 40$ for **Playground** and **Mining** domain, and linearly annealed the temperature from $T = 1$ to $T = 40$ during adaptation phase for **SC2LE** domain. Intuitively speaking, we act more confidently (*i.e.*, higher temperature T) as we collect more data since the inferred subtask graph will become more accurate.

C DERIVATION OF MULTI-TASK SUBTASK GRAPH INFERENCE

Let τ be the adaptation trajectory of the current task $\mathcal{M}_G$, and $\mathcal{T}^{\text{p}} = \{\tau_1^{\text{p}}, \ldots, \tau_{|\mathcal{T}^{\text{p}}|}^{\text{p}}\}$ be the adaptation trajectories of the seen training tasks.

Then, from Bayesian rule, we have

$$\begin{aligned}
\pi(o|s, \tau, \mathcal{T}^{\text{p}}) &= \textstyle\sum_G \pi(o|s, G) p(G|\tau, \mathcal{T}^{\text{p}}) && (29)\\
&\propto \textstyle\sum_G \pi(o|s, G) p(\tau|G, \mathcal{T}^{\text{p}}) p(G|\mathcal{T}^{\text{p}}) && (30)\\
&= \textstyle\sum_G \pi(o|s, G) p(\tau|G) p(G|\mathcal{T}^{\text{p}}) && (31)\\
&\propto \textstyle\sum_G \pi(o|s, G) p(\tau|G) p(\mathcal{T}^{\text{p}}|G) p(G), && (32)
\end{aligned}$$

where Equation (31) holds because τ and $\mathcal{T}^{\text{p}}$ are independently observed variables. Since summing over all G is computationally intractable, we instead approximate it by computing the sample estimates of π. Specifically, we compute the policy $\pi(o|s, G)$ at the maximum likelihood estimates (MLE) of G for prior and current tasks, that is $G^\tau = \arg\max_G p(\tau|G)$ and $G^{\text{p}} = \arg\max_G p(\mathcal{T}^{\text{p}}|G)$ respectively, and combine them with the weight α:

$$\pi(o|s, \tau, \mathcal{T}^{\text{p}}) \simeq \pi(o|s, G^\tau)^\alpha \pi(o|s, G^{\text{p}})^{(1-\alpha)}. \tag{33}$$

Finally, we deploy the GRProp policy as a contextual policy:

$$\pi^{\text{eval}}(\cdot|\tau, \mathcal{T}^{\text{p}}) \simeq \text{GRProp}(\cdot|G^\tau)^\alpha \text{GRProp}(\cdot|G^{\text{p}})^{(1-\alpha)}. \tag{34}$$

D PSEUDO-CODE OF OUR ALGORITHM

The Algorithm 3 below describes the pseudo-code of the meta-training process of our algorithm.

Algorithm 3 Meta-training: learning the prior

Require: Adaptation policy π^{adapt}
Ensure: Prior set $\mathcal{T}^{\text{p}}$

```
1: T^p ← ∅
2: for each task M ∈ M^train do
3:     Rollout adaptation policy:
       τ = {s_t, o_t, r_t, d_t}_{t=1}^K ~ π^adapt in task M
4:     Infer subtask graph G^τ = arg max_G p(τ|G)
5:     π^eval = GRProp(G^τ)
6:     Evaluate the agent: τ^eval ~ π^eval in task M
7:     Update prior T^p ← T^p ∪ (G^τ, τ)
8: end for
```

E EXTENDED RELATED WORK

Multi-task reinforcement learning. Multi-task reinforcement learning aims to learn an inductive bias that can be shared and used across a variety of related RL tasks to improve the task generalization. Early works mostly focused on the transfer learning oriented approaches (Lazaric, 2012; Taylor & Stone, 2009) such as instance transfer (Lazaric et al., 2008) or representation transfer (Konidaris & Barto, 2006). However, these algorithms rely heavily on the prior knowledge about the allowed task differences. Hausman et al. (2018); Pinto & Gupta (2017); Wilson et al. (2007) proposed to train a multi-task policy with multiple objectives from different tasks. However, the gradients from different tasks may conflict and hurt the training of other tasks. To avoid gradient conflict, Zhang & Yeung (2014); Chen et al. (2018); Lin et al. (2019) proposed to explicitly model the task similarity. However, dynamically modulating the loss or the gradient of RL update often results in the instability in optimization. Our multi-task learning algorithm also takes the transfer learning oriented viewpoint; MTSGI captures and transfers the task knowledge in terms of the subtask graph. However, our work does not make a strong assumption on the task distribution. We only assume that the task is parameterized by unknown subtask graph, which subsumes many existing compositional tasks (*e.g.*, Oh et al. (2017); Andreas et al. (2017); Huang et al. (2018), etc).

Extended - web navigating RL agent. Previous work introduced MiniWoB (Shi et al., 2017) and MiniWoB++ (Liu et al., 2018) benchmarks that are manually curated sets of simulated toy environments for the web navigation problem. They formulated the problem as acting on a page represented as a Document Object Model (DOM), a hierarchy of objects in the page. The agent is trained with human demonstrations and online episodes in an RL loop. Jia et al. (2019) proposed a graph neural network based DOM encoder and a multi-task formulation of the problem similar to this work. Gur et al. (2018) introduced a manually-designed curriculum learning method and an LSTM based DOM encoder. DOM level representations of web pages pose a significant sim-to-real gap as simulated websites are considerably smaller (100s of nodes) compared to noisy real websites (1000s of nodes). As a result, these models are trained and evaluated on the same simulated environments which is difficult to deploy on real websites. Our work formulates the problem as abstract web navigation on real websites where the objective is to learn a latent subtask dependency graph similar to sitemap of websites. We propose a multi-task training objective that generalizes from a fixed set of real websites to unseen websites without any demonstration, illustrating an agent capable of navigating real websites for the first time.

Planning Approaches for Compositional Task Previous work has tackled the compositional tasks using the Hierarchical Task Network (HTN) planning (Sacerdoti, 1975b;a; Tate, 1977) in a (single) goal-conditioned RL setting. The HTN allows the agent to reason the tasks at multiple levels of abstraction, when rich knowledge at those abstraction levels are available. Specifically, HTN models the primitive tasks (or the *subtasks* in our terminology) by the precondition and the effects, and aim to find the sequence of actions (or the *options* in our terminology) that execute each subtasks via planning on the HTN. They aim to execute a single goal task, often with assumptions of simpler subtask dependency structures (Ghazanfari & Taylor, 2017; Liu et al., 2016) such that the task structure can be constructed from the successful trajectories. Also, they often require expensive searching to find the solution. In contrast, we tackle a more general and challenging setting, where each subtask gives a reward (*i.e.*, multi-goal setting) and the goal is to maximize the cumulative sum of reward within an episode. Moreover, we avoid any expensive searching and propose to use neural network to directly map the task structure into policy. Lastly, we aim to achieve zero-/few-shot task generalization, which is not achievable with the HTN methods since they require the full specification of the action models in the testing.

F ADDITIONAL EXPERIMENT RESULTS

F.1 FULL EXPERIMENT RESULTS ON THE PERFORMANCE OF THE AGENTS ON 20 WEBSITES IN *SymWoB*

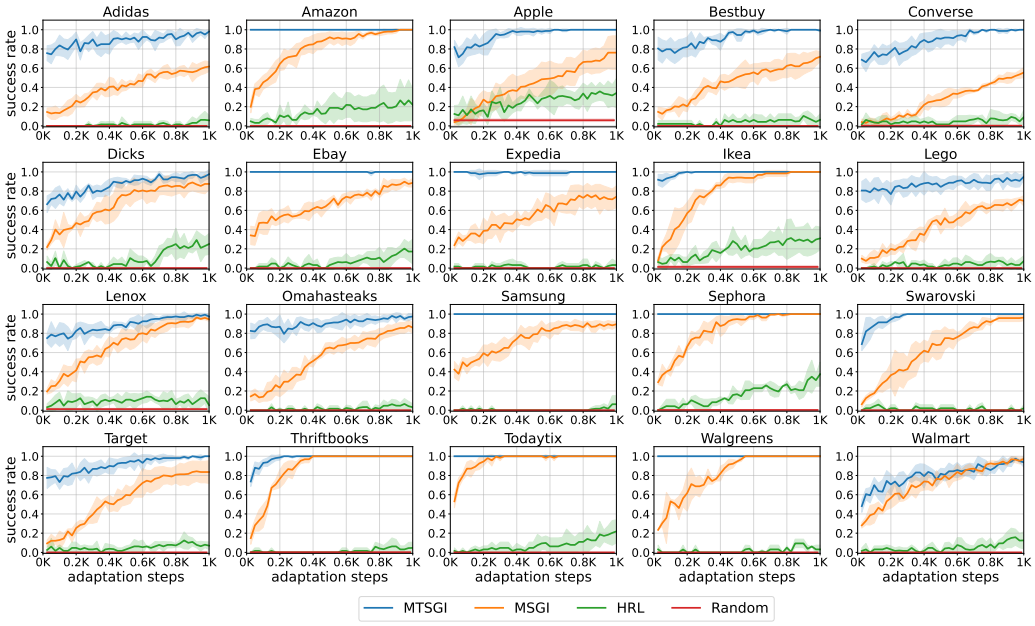

Figure 9: Results of the success rate for compared methods in the test phase with respect to the adaptation steps on 20 environments in *SymWoB* domain.

F.2 VISUALIZATION OF THE MULTI-TASK SUBTASK GRAPH INFERENCE PROCESS

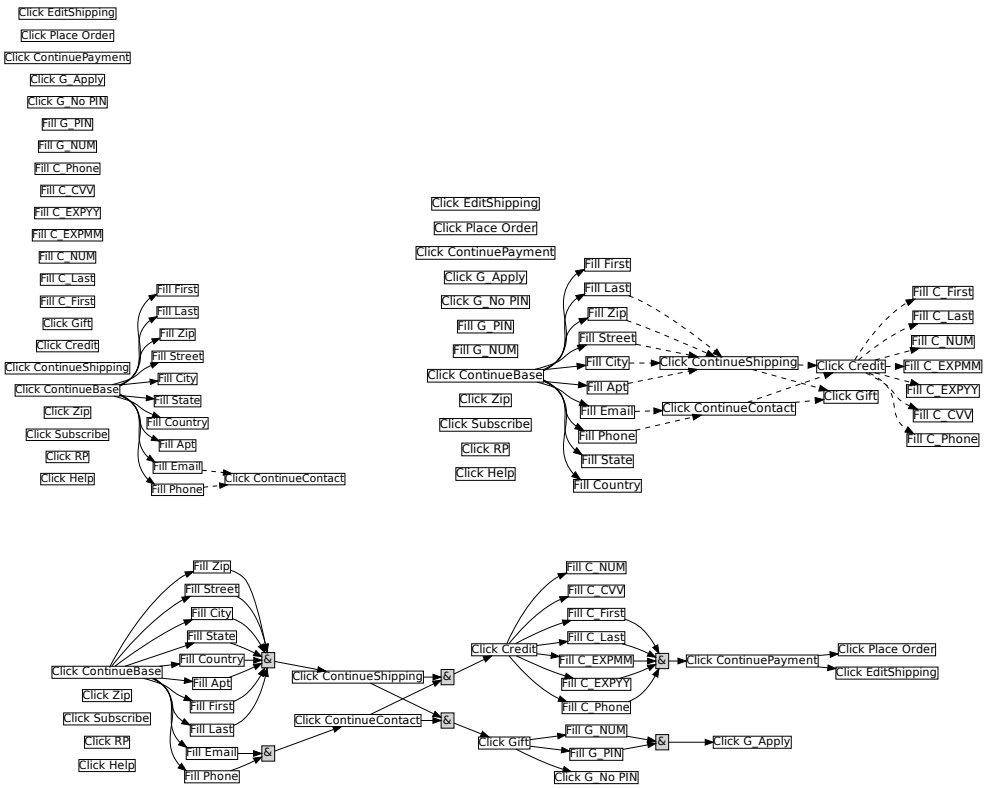

Figure 10: The subtask graphs inferred by our MTSGI with varying adaptation steps on **Walmart** domain: 0 steps (*Top, Left*), 400 steps (*Top, Right*), and 1000 steps (*Bottom*).

Figure 10 qualitatively shows how the multi-task subtask graph inference proceeds over the adaptation. In the beginning (*Top, Left*), the agent has only prior information and the prior provides a partial information about the subtask graph, which is the preconditions in the first webpage. As the agent further explores the webpage (*Top, Right*), the subtask graph gets more accurate, but due to insufficient exploration, there are many missing preconditions, especially for the subtasks in the last webpage. After sufficient adaptation (*Bottom*), our MTSGI can infer quite accurate subtask graph compared to the ground-truth subtask graph in Figure 11; *i.e.*, only missing two preconditions.

F.3 FULL EXPERIMENT RESULTS ON THE QUALITATIVE EVALUATION OF THE INFERRED SUBTASK GRAPH

Figures 11-30 qualitatively evaluate the task inference of our MTSGI by comparing the inferred subtask graph and the ground-truth. It is clear from the figure that the inferred subtask graphs have only a small portion of missing or redundant edges, while most of the nodes and edges are the same as ground-truth graph.

G DETAILS ON *SymWoB* DOMAIN

Subtask Graph Setting										
Task	**Adidas**	**Amazon**	**Apple**	**BestBuy**	**Converse**	**Dick's**	**eBay**	**Expedia**	**Ikea**	**Lego**
#Subtasks	46	31	43	37	42	39	39	36	39	45
#Distractors	8	4	5	6	6	6	5	5	5	6
Episode length	41	27	40	37	43	37	37	40	37	37
Task	**Lenox**	**Omahasteaks**	**Samsung**	**Sephora**	**Swarovski**	**Target**	**Thriftbooks**	**Todaytix**	**Walgreens**	**Walmart**
#Subtasks	45	44	42	49	45	39	43	23	38	46
#Distractors	4	6	6	7	7	6	8	3	7	5
Episode length	41	38	41	45	38	37	33	20	50	43

Table 1: The task configuration of the tasks in *SymWoB* domain. Each task is parameterized by different subtask graphs, and the episode length is manually set according to the challengeness of the tasks.

In this paper, we introduce the *SymWoB* domain, which is a challenging symbolic environment that aims to reflect the hierarchical and compositional aspects of the checkout processes in the real-world websites. There are total 20 different *SymWoB* websites that are symbolic implementations of the actual websites: **Adidas**, **Amazon**, **Apple**, **BestBuy**, **Converse**, **Dick's**, **eBay**, **Expedia**, **Ikea**, **Lego**, **Lenox**, **Omahasteaks**, **Samsung**, **Sephora**, **Swarovski**, **Target**, **Thriftbooks**, **Todaytix**, **Walgreens**, and **Walmart**. All of these websites are generated by analyzing the corresponding real websites and reflecting their key aspects of checkout process. The main goal of each website is to navigate through the web pages within the website by clicking and filling in the web elements with proper information which leads to the final web page that allows the agent to click on the `Place_Order` button, which indicates that the agent has successfully finished the task of checking out.

G.1 IMPLEMENTATION DETAIL

In this section, we describe the detailed process of implementing an existing website into a symbolic version. We first fill out the shopping cart with random products, and we proceed until placing the order on the actual website. During the process, we extract all the interactable web elements on the webpage. We repeat this for all the websites and form a shared subtask pool where similar web elements in different websites that have same functionality are mapped to the same subtask in the shared subtask pool. Then, we extract the precondition relationship between subtasks from the website and form the edges in the subtask graph accordingly. Finally, we implement the termination condition and the subtask reward to the failure distractor (See Appendix G.3) and the goal subtasks.

G.2 COMPARISON OF THE WEBSITES

The agent's goal on every website is the same, that is placing the checkout order. However, the underlying subtask graphs, or task structure, of the websites are quite diverse, making the task much more challenging for the agent. Figures 11-30 visualize the ground truth subtask graph of all the websites. One of the major sources of diversity in subtask graphs is in the various ordering of the web pages. In a typical website's checkout process, some of the most common web pages include the shipping, billing, and payment web pages, each of which has a collection of corresponding subtasks. In Figure 11, for example, the shipping web page is represented by the collection of the subtasks on the left side from `Fill_Zip` to `Fill_Last` and `Click_ContinueShipping`, and these come *before* the payment web page that is represented by the subtasks on the right side from `Click_Credit` to `Click_ContinuePayment`. On the other hand, in Figure 17 and Figure 16, the similar web pages are either connected in a different ordering, from payment to shipping web page, or placed on the same line side by side. Since the web pages can vary on how they are ordered, it allows the subtask graphs to have a variety of shapes such as deep and narrow as in Figure 19 or wide and shallow as in Figure 16. Different shape of the subtask graphs means different precondition between the tasks, making it non-trivial for the agent to transfer its knowledge about one to the other.

Another major source of diversity is the number of web elements in each web page. Let's compare the web elements of the shipping web page in Figure 13 and Figure 14. These are the subtasks

that are connected to `Click_ContinueShipping` and as well as itself. We can see that the two websites do not have the same number of the web elements for the shipping web pages: the **Converse** website requires more shipping information to be filled out than the **Dick's** website. Such variety in the number of web elements, or subtasks, allows the subtask graphs of the websites to have diverse preconditions as well.

G.3 DISTRACTOR SUBTASKS

In addition to the different task structures among the websites, there are also *distractor* subtasks in the websites that introduces challenging components of navigating the real-world websites. There are two different types of distractor subtasks: the one that terminates the current episode with a negative reward and the another one that has no effect. The former, which we also call it the *failure* distractor subtask, represents the web elements that lead the agent to some external web pages like `Help` or `Terms_of_Use` button. The latter is just called the distractor subtask, where executing the subtask does not contribute to progressing toward the goal (*e.g.*, `Click EditShipping` subtask in **Converse**). Each website has varying number of distractor subtasks and along with the shallowness of the task structure, the number of distractor subtasks significantly affects the difficulty of the task.

H DETAILS OF THE AGENT IMPLEMENTATION

We implement all of our algorithms, including both MTSGI and the baselines, on top of the recently introduced RL framework called Acme (Hoffman et al., 2020).

H.1 MSGI

Similar to the MTSGI agent, the MSGI agent uses the soft-version of UCB exploration policy as the adaptation policy, instead of its original hard-version due to a better performance. Furthermore, MSGI also uses inductive logic programming (ILP) in order to infer the subtask graph of the current task. But unlike the MTSGI agent that learns prior across the tasks through multi-task learning, the MSGI agent does not exploit the subtask graphs inferred from the previous tasks.

H.2 HRL

The hierarchical RL (HRL) agent is an option-based agent that can execute temporally extended actions. For the *Mining* domain, where there is a spatial component in the observation input, we use convolutional neural network (CNN) in order to encode the spatial information and get concatenated along with the other additional inputs, which are encoded using fully-connected (FC) networks. The concatenated embedding then gets passed on to GRU, which is followed by two separate heads for value and policy function outputs.

The details of the HRL architecture for *Mining* domain are: Conv1(16x1x1-1)-Conv2-(32x3x3-1)-Conv3(64x3x3-1)-Conv4(32x3x3-1)-Flatten-FC(256)-GRU(512). The HRL architecture for *SymWoB* domain is almost identical except it replaces the CNN module with fully-connected layers for processing non-spatial information: FC(64)-FC(64)-FC(50)-FC(50)-GRU(512). We use ReLU activation function in all the layers.

For training, we use multi-step actor-critic (A2C) in order to train HRL. We use multi-step learning of $n = 10$, learning rate 0.002, entropy loss weight of 0.01 and critic loss weight of 0.5. We use RMSProp optimizer to train the networks, where its decay rate is set to 0.99 and epsilon to 0.00001. We also clip the gradient by setting the norm equal to 1.0. Most importantly, the network parameters of the HRL agent gets reset for every new task since HRL is not a meta-RL agent.

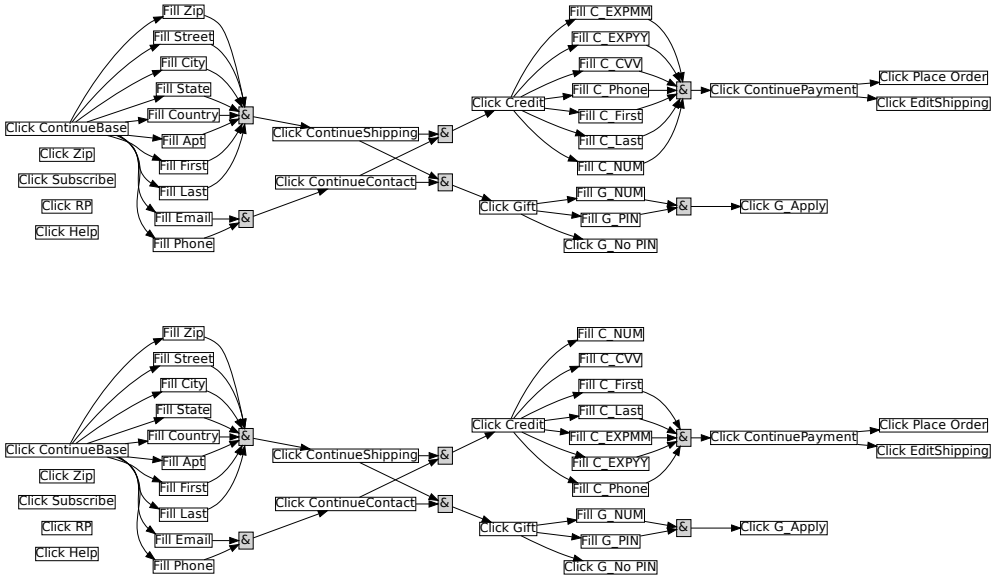

Figure 11: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **Walmart** website.

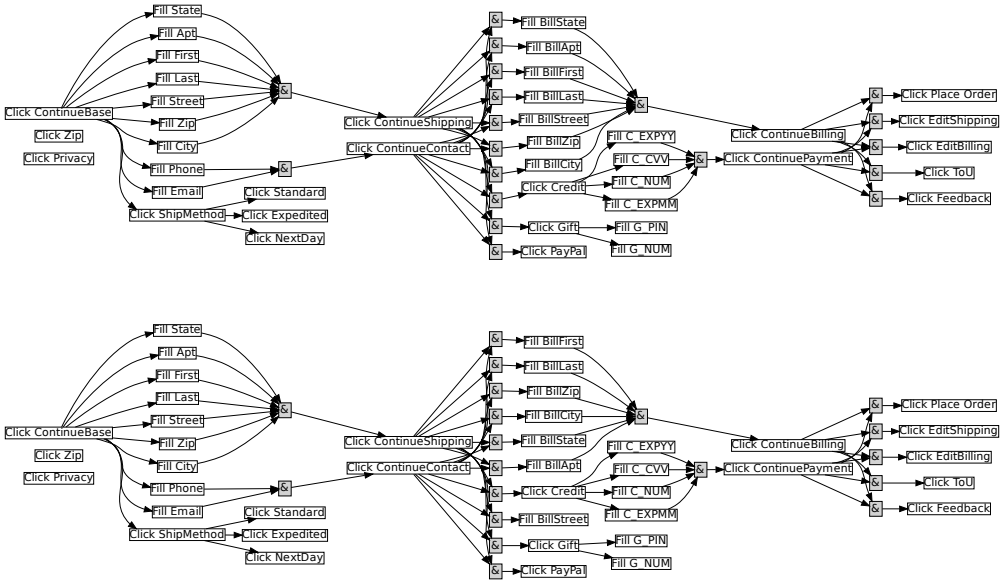

Figure 12: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **Converse** website.

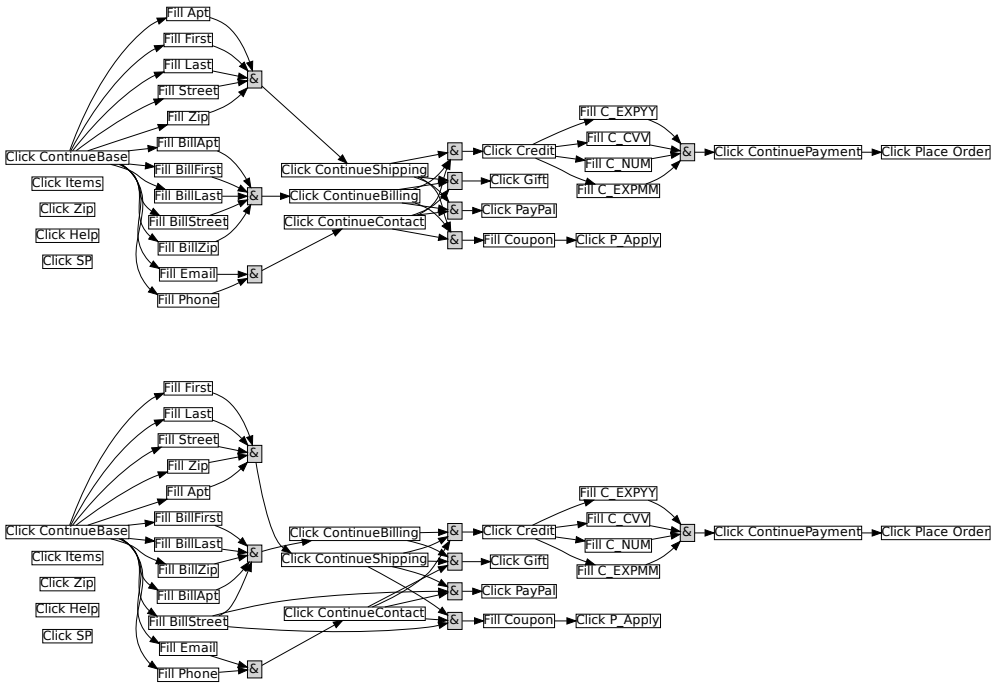

Figure 13: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **Dick's** website.

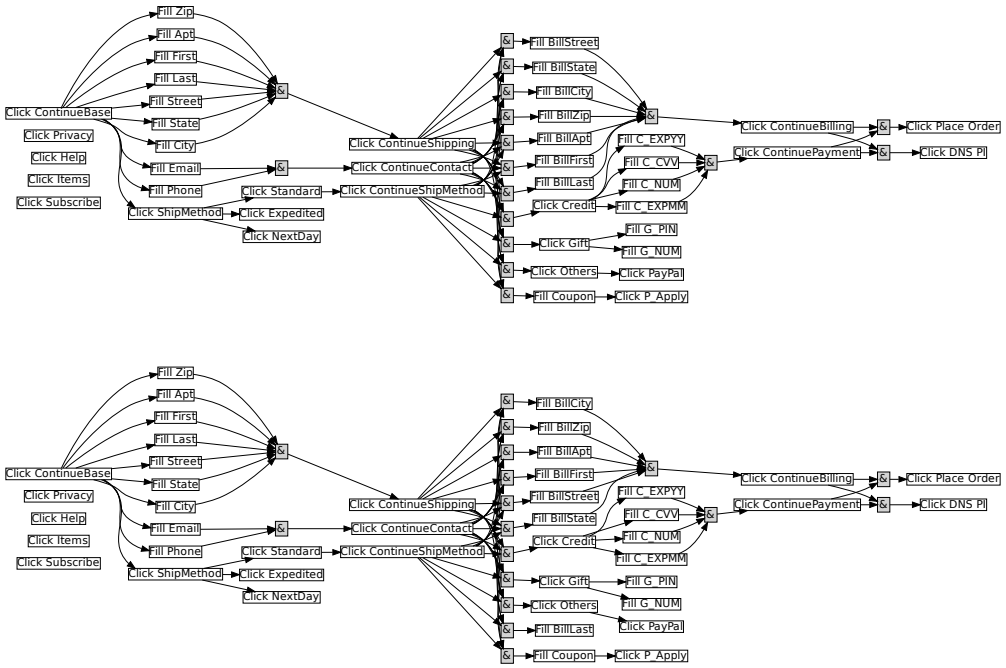

Figure 14: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **BestBuy** website.

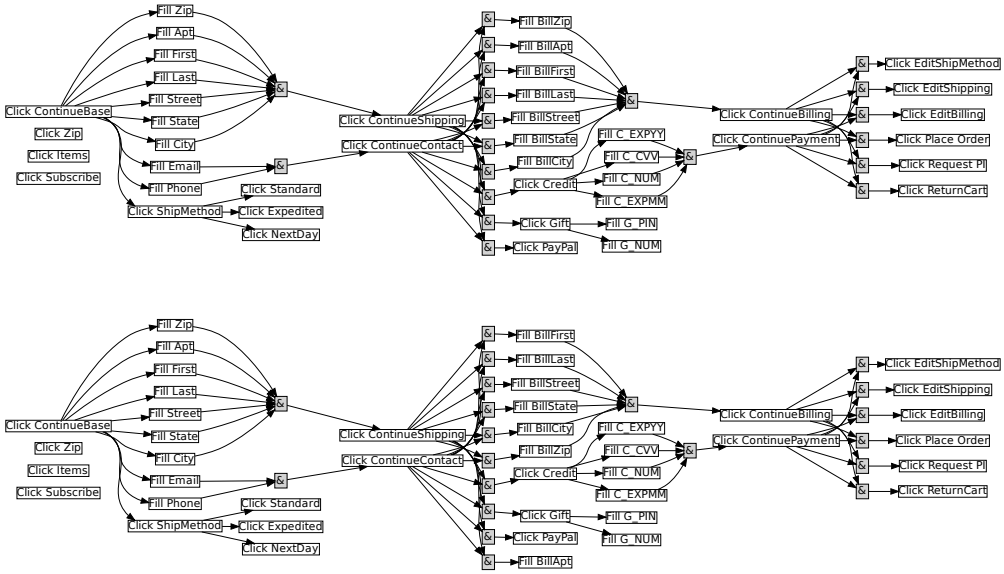

Figure 15: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **Apple** website.

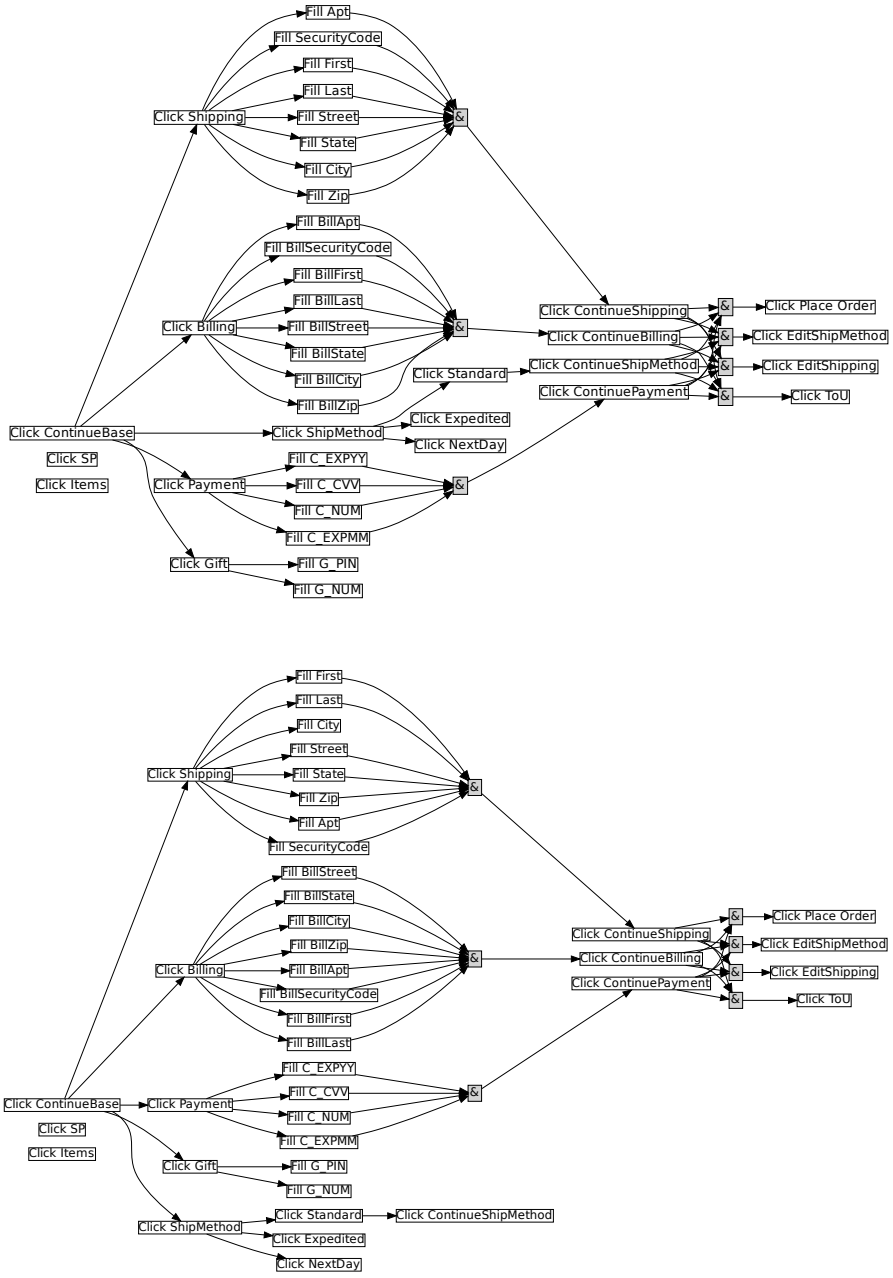

Figure 16: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **Amazon** website.

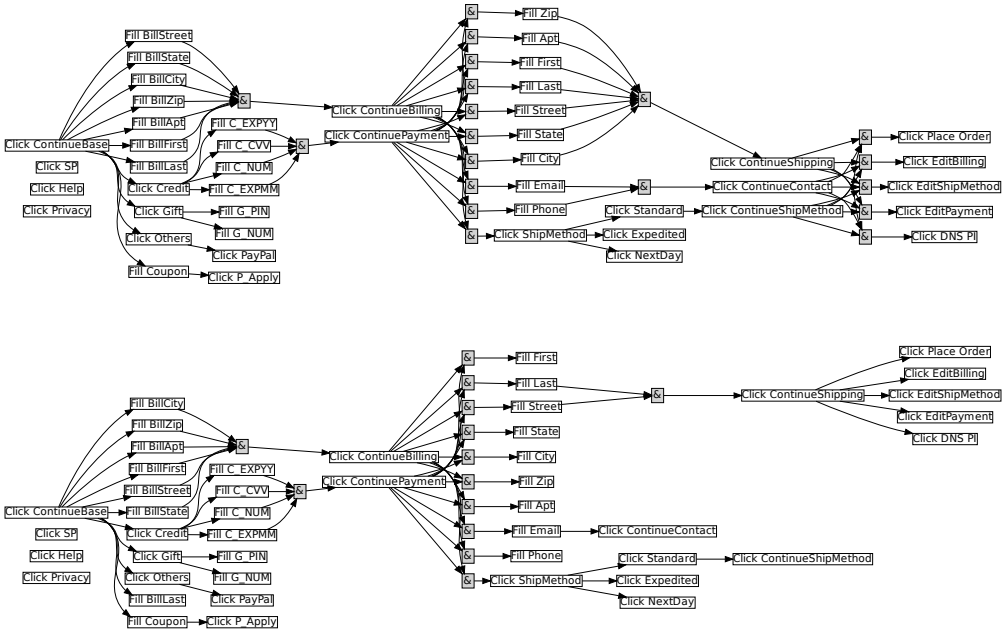

Figure 17: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **Samsung** website.

Figure 18: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **eBay** website.

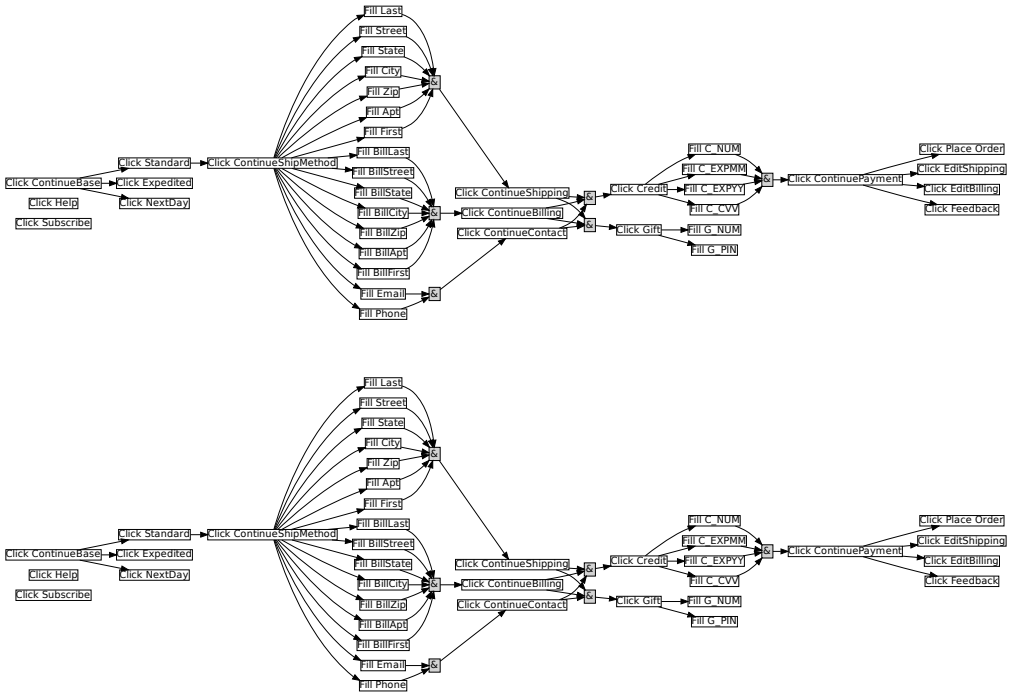

Figure 19: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **Ikea** website.

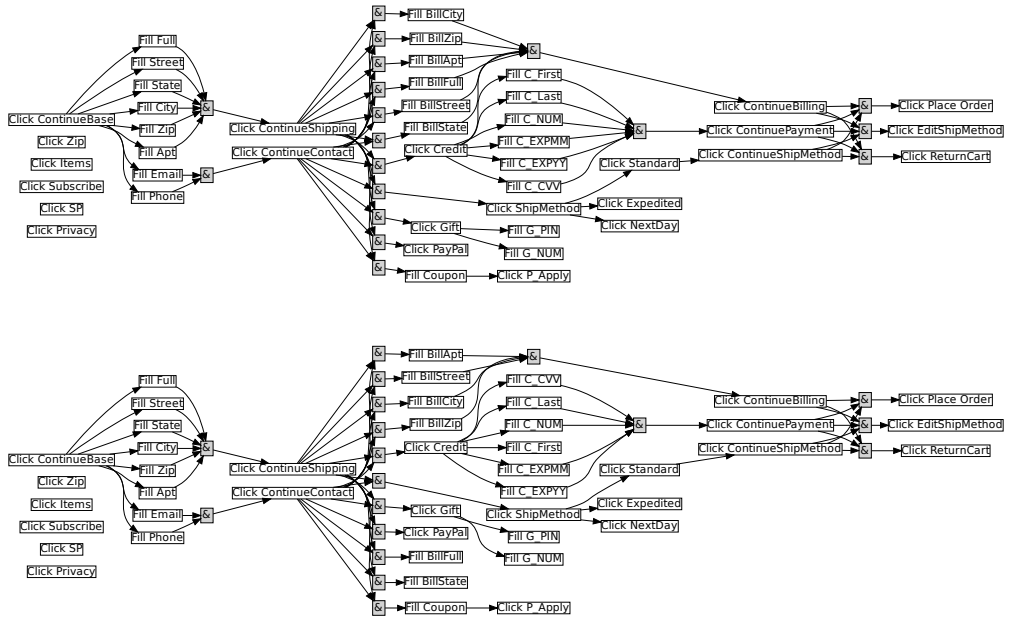

Figure 20: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **Target** domain.

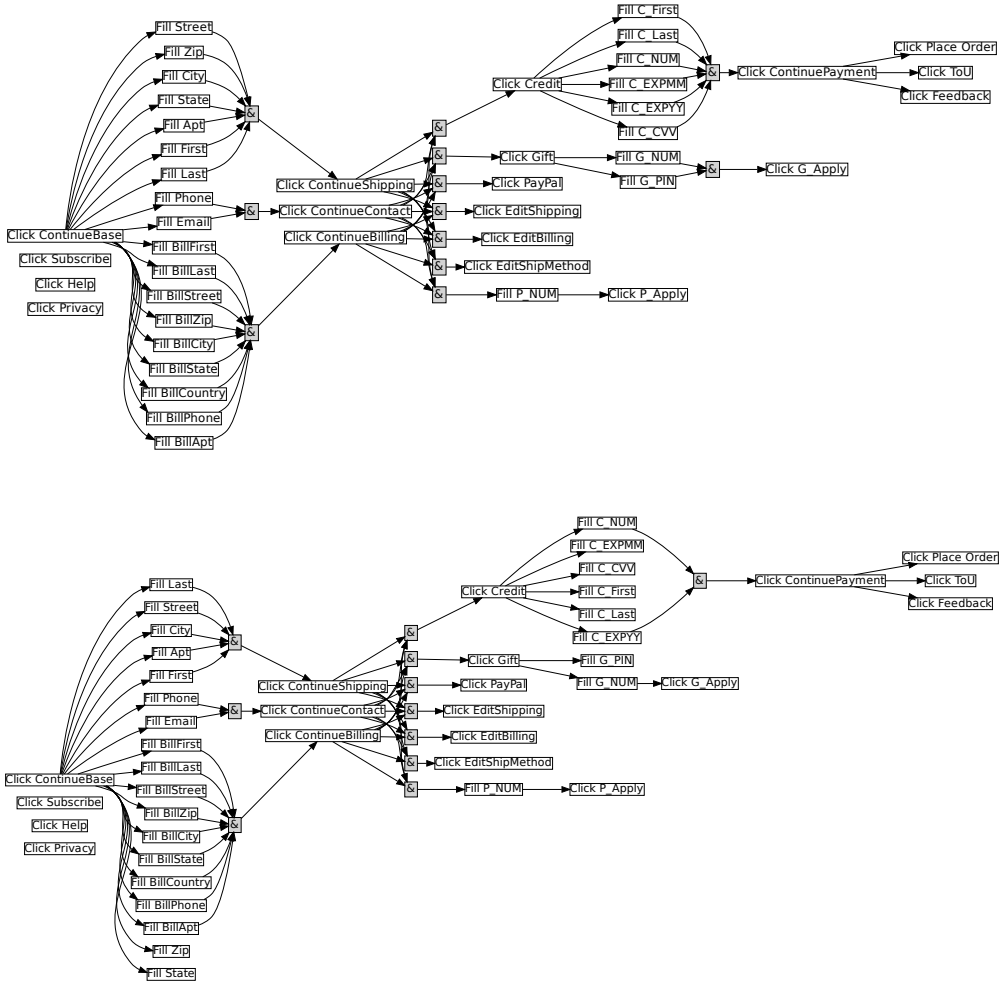

Figure 21: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **Adidas** domain.

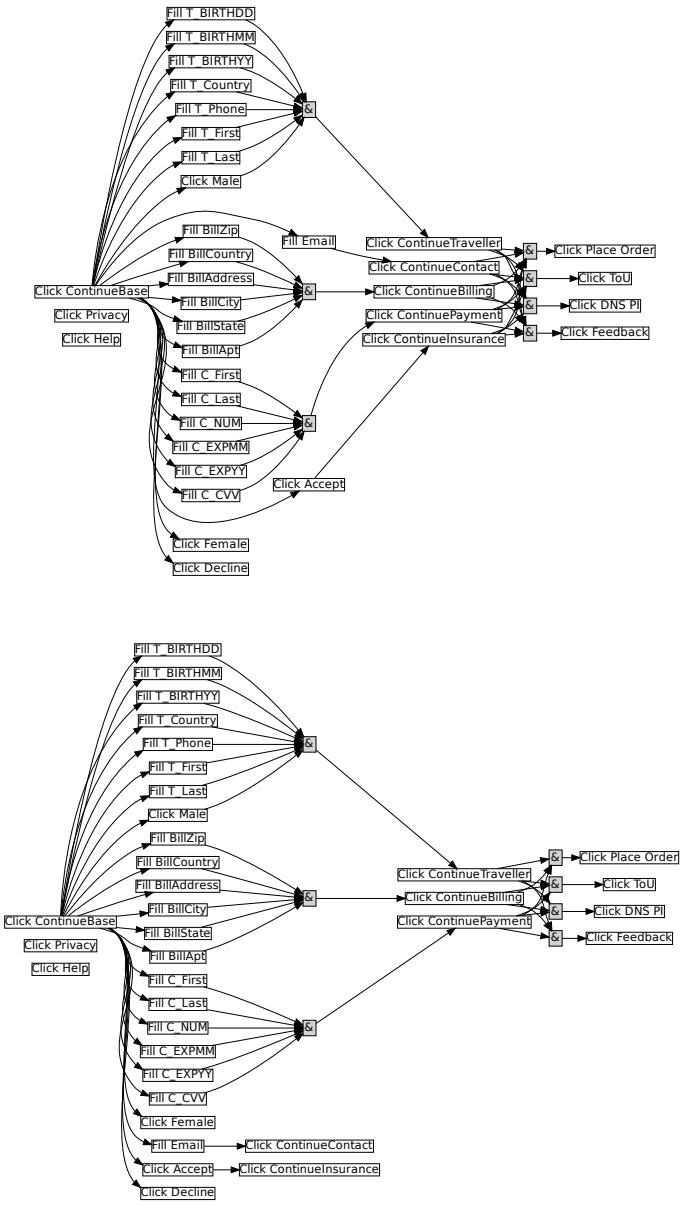

Figure 22: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **Expedia** domain.

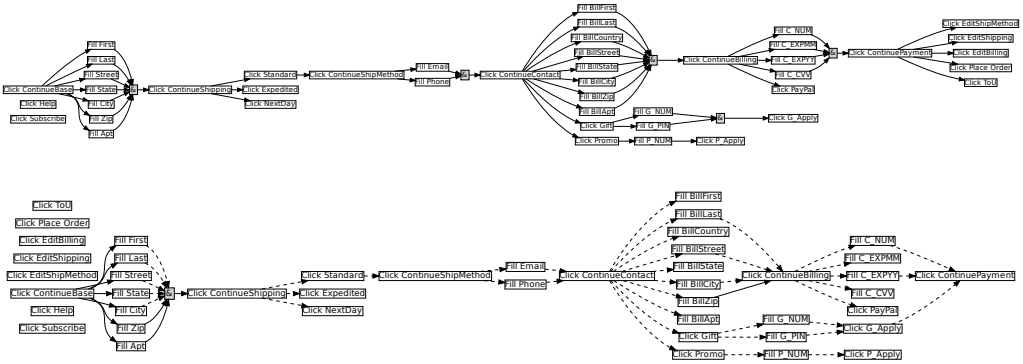

Figure 23: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **Lego** domain.

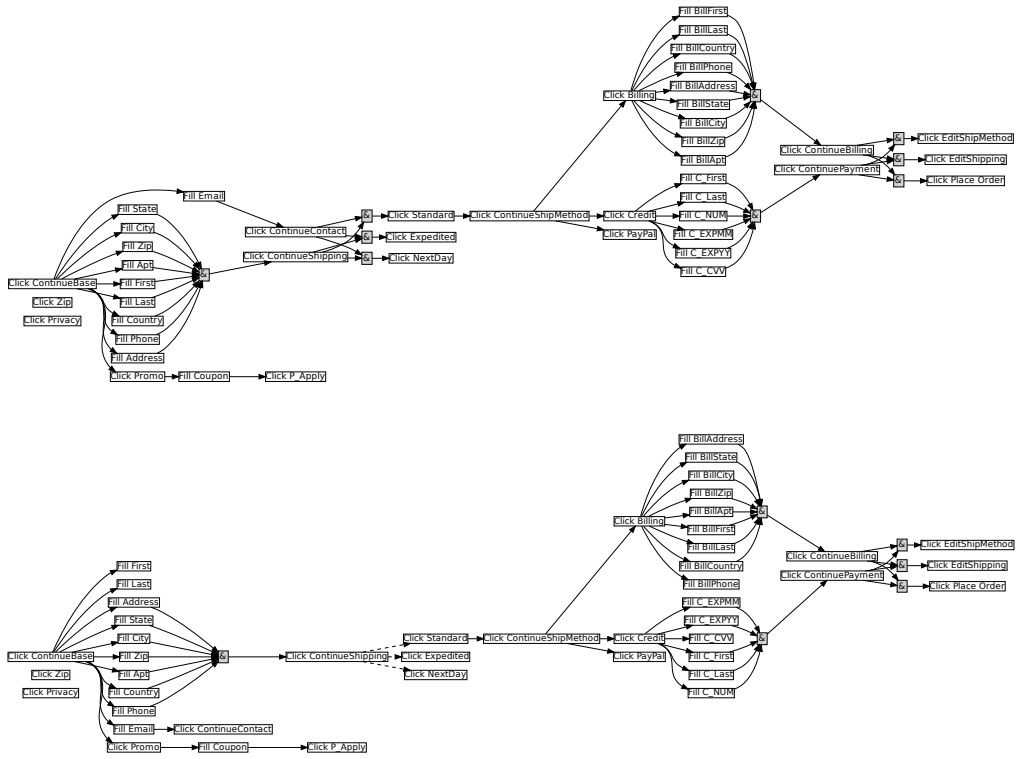

Figure 24: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **Lenox** domain.

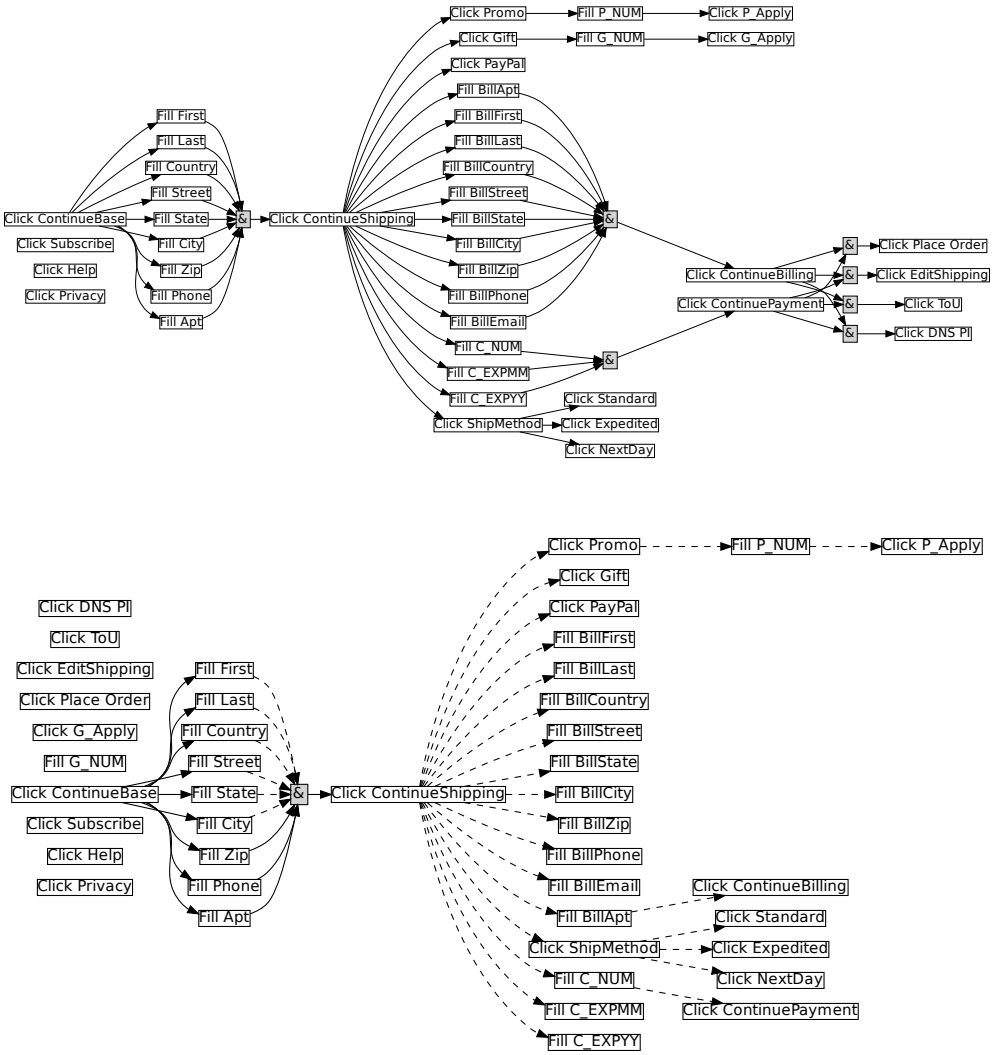

Figure 25: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **Omahasteaks** domain.

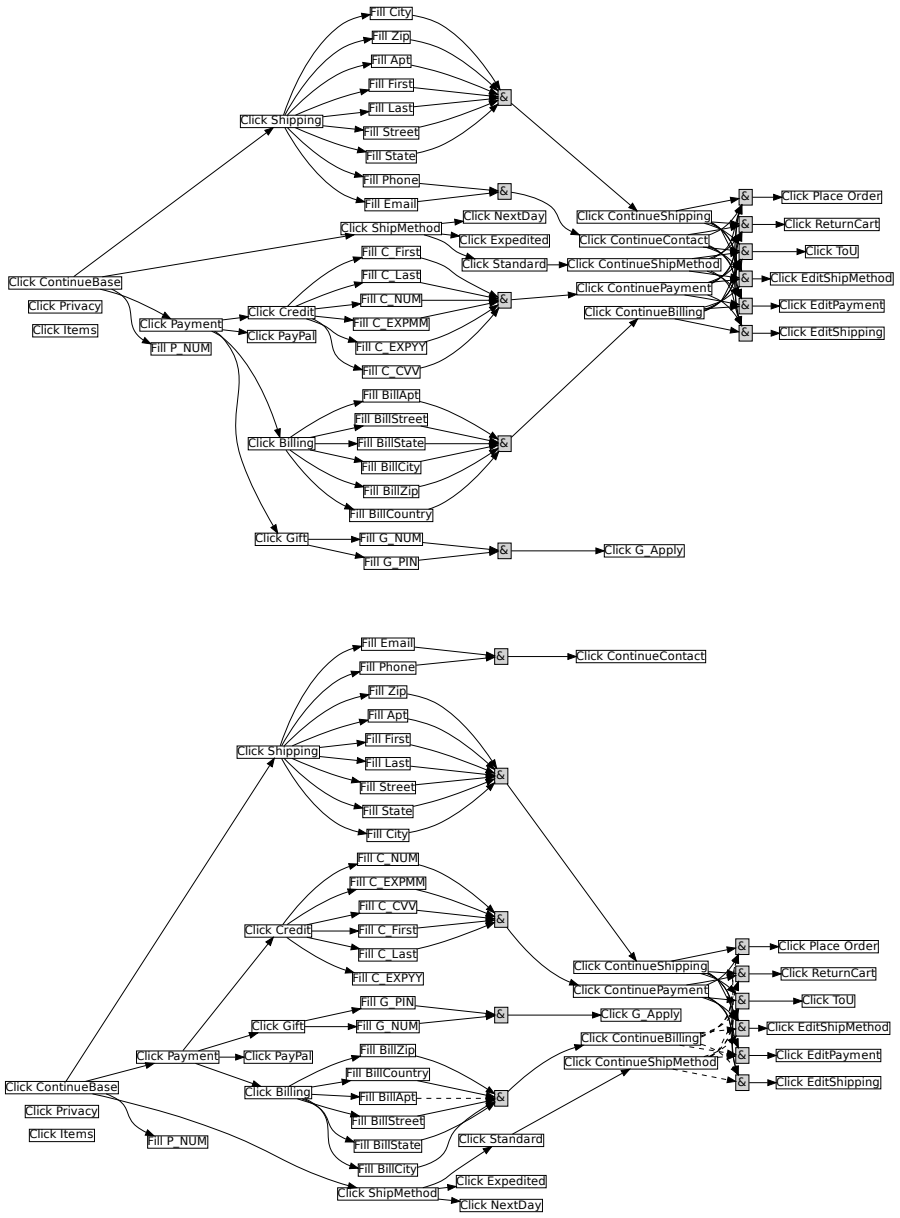

Figure 26: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **Sephora** domain.

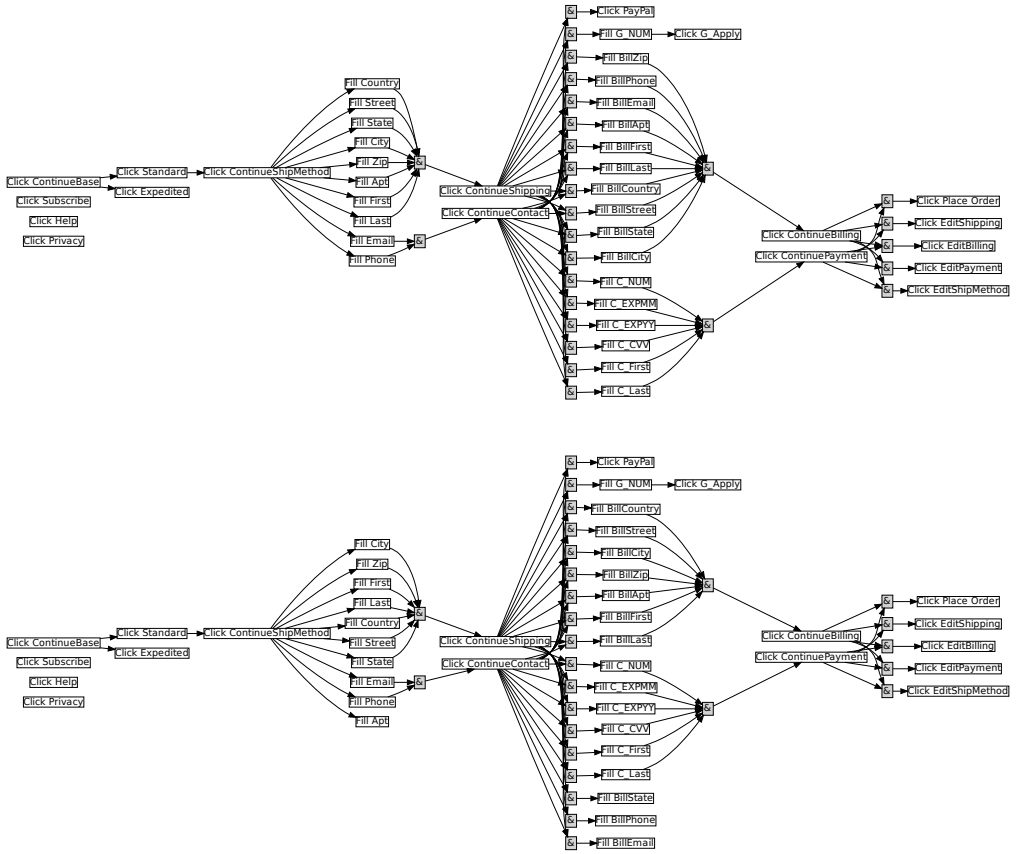

Figure 27: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **Swarovski** domain.

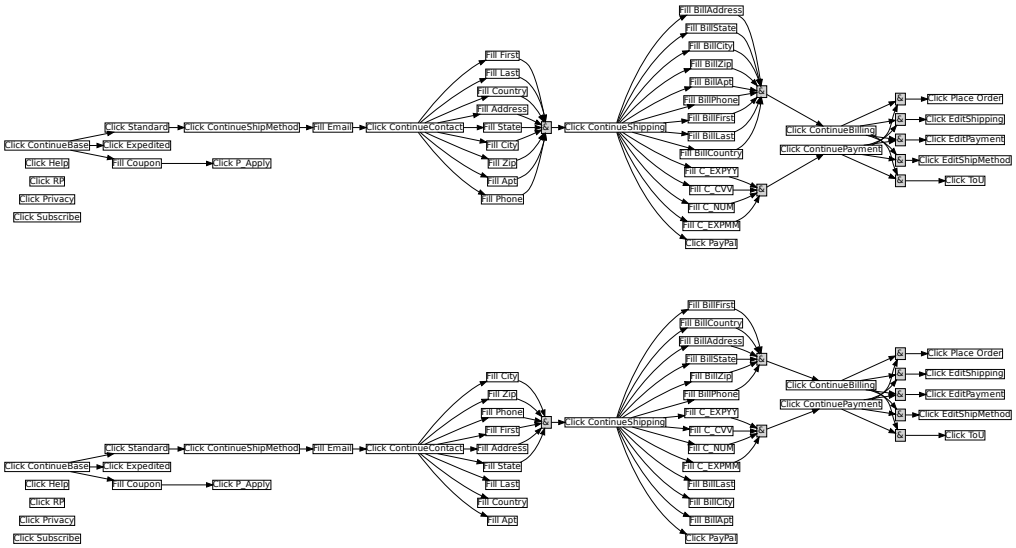

Figure 28: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **Thriftbooks** domain.

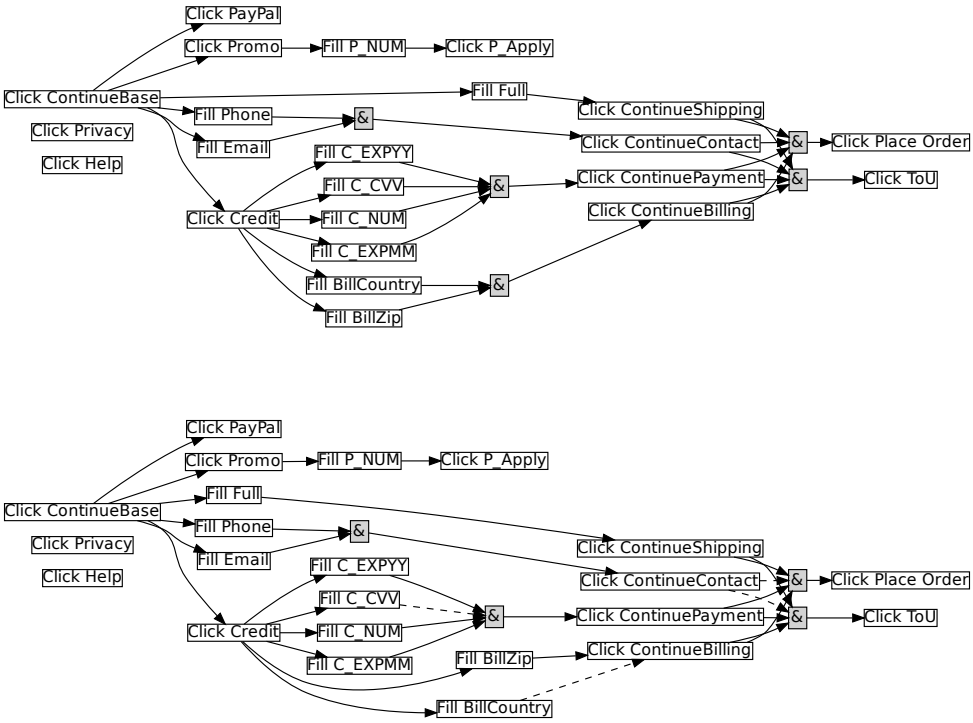

Figure 29: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **Todaytix** domain.

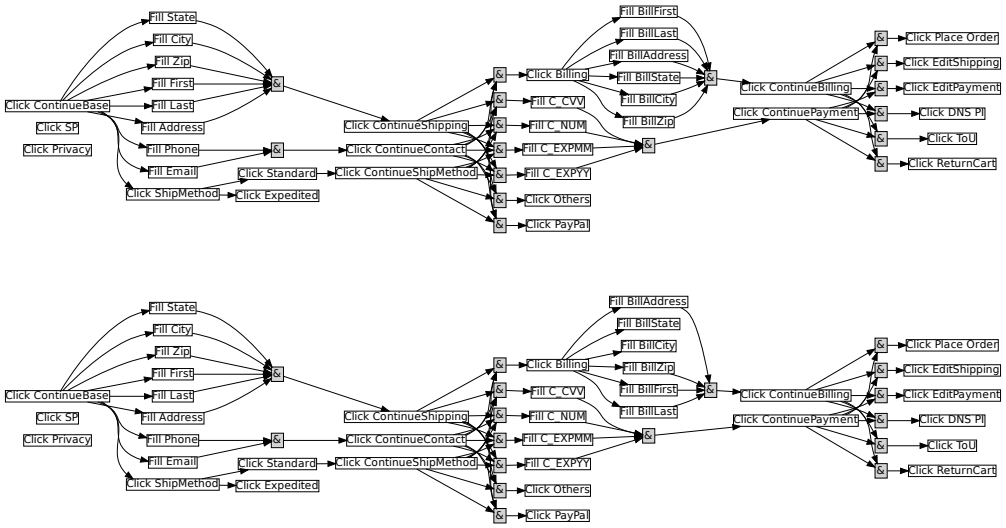

Figure 30: *(Top)*The ground-truth and *(Bottom)*the inferred subtask graphs of **Walgreens** domain.