5 Appendix

5.1 Play Environment and Tasks

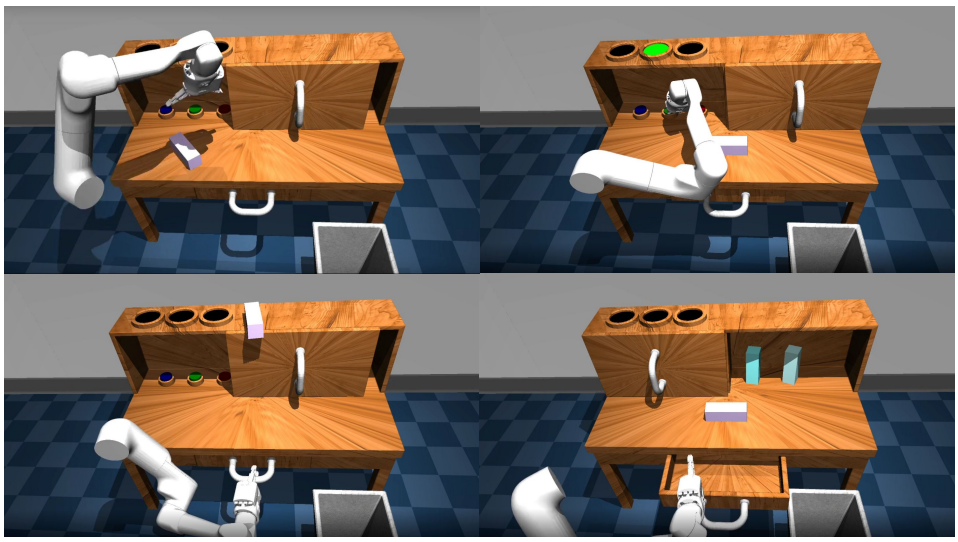

Figure 6: The Playground 3D robotic tabletop environment used for data collection and evaluation.

The playground environment used in this work is shown in Fig. 6. Significantly more detail can be found in [38]. It contains the following set of tasks used within chains (examples are visualized in Fig. 7):

- Grasp lift: Grasp a block out of an open drawer and place it on the desk surface.
- Grasp upright: Grasp an upright block off of the surface of the desk and lift it to a desired position.
- Grasp flat: Grasp a block lying flat on the surface of the desk and lift it to a desired position.
- Open sliding: Open a sliding door from left to right.
- Close sliding: Close a sliding door from right to left.
- Drawer: Open a closed desk drawer.
- Close Drawer: Close an open desk drawer.
- Sweep object: Sweep a block from the desk into a drawer.
- Knock object: Knock an upright object over.
- Push red button: Push a red button inside a desk shelf.
- Push green button: Push a green button inside a desk shelf.
- Push blue button: Push a blue button inside a desk shelf.
- Rotate left: Rotate a block lying flat on the table 90 degrees counter clockwise.
- Rotate right: Rotate a block lying flat on the table 90 degrees clockwise.
- Sweep left: Sweep a block lying flat on a table a specified distance to the left.
- Sweep right: Sweep a block lying flat on a table a specified distance to the right.
- Put into shelf: Place a block on a table into a shelf.
- Pull out of shelf: Retrieve a block from a shelf and put on the table.

The full state space is 19-dimensional and contains:

- a robot arm and gripper (8 dimensional position, orientation, and the amount each finger is closed)
- a block (6 dimensional position and orientation)
- a sliding door (1 dimensional continuous position)

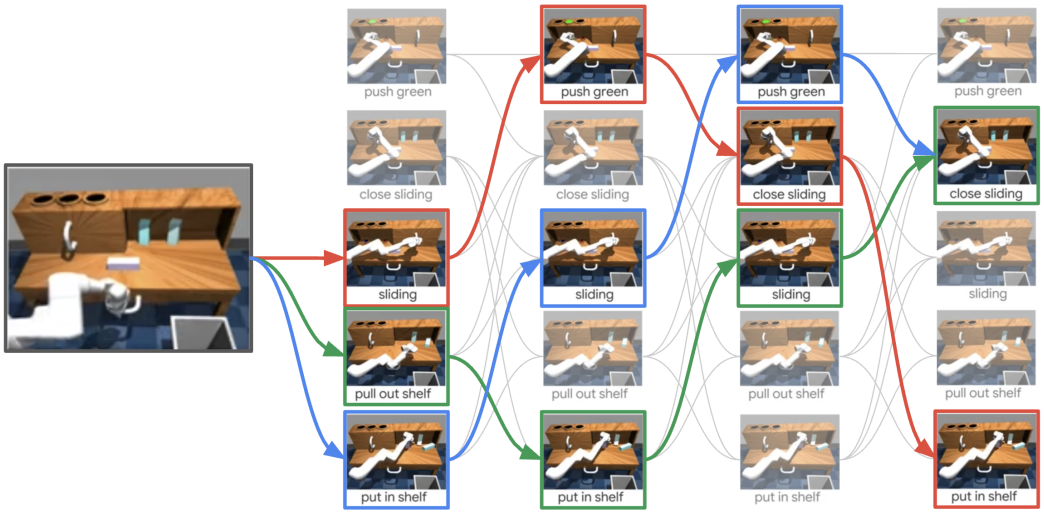

Figure 7: BELT is evaluated on sequences of chained short-horizon tasks generated in a feasible order.

- a drawer door (1 dimensional continuous position)
- three buttons (1 dimensional each, continuously tracking the amount pressed)

5.2 Algorithmic Parameters

Networks: The policy is an RNN with 2 hidden layers of size 2048 each, mapping inputs to the parameters of MODL distribution on quantized actions. The Temporal Distance Classifier (TDC) and model networks all have the same architecture, a feedforward network with 2 hidden layers of size 2048 each. The TDC and models were trained on demonstration data for $\sim$ 100k rollouts.

BELT Parameters: BELT was run for 2500 samples drawn from demonstration data of the policy rollouts, rather than the play data generated by human operators. We note that we also tried samples from the play dataset, but found many of the goals unreachable by the policy, and overall worse performance. The edges were rolled out with a randomly drawn timestep of $T \in \{32, 64, 96\}$, as inspired by [10]. Each plan was replayed nine times to study robustness.

5.3 Biasing the Search

In this section we examine the best techniques for biasing the tree search. We examined performance along the axes of solve rate and solution cost, mirroring the axes of exploration of new paths and exploitation of good current paths. The results are shown in Fig. 8 and the bias method is shown in Fig. 4a. As the TDC radius increases the solve rate decreases along with the cost; this is a result of lower cost paths being built off more often, but less exploration of longer paths. We note that this lower cost is both due to lower cost paths being more often selected and due to simpler problems being solved, so the effect on cost becomes exaggerated. As the TDC radius decreases it approaches the performance of random sampling. The success rate peaks $d_{\text{cutoff}} = 64$, outperforming L2 and random in solve rate and cost.

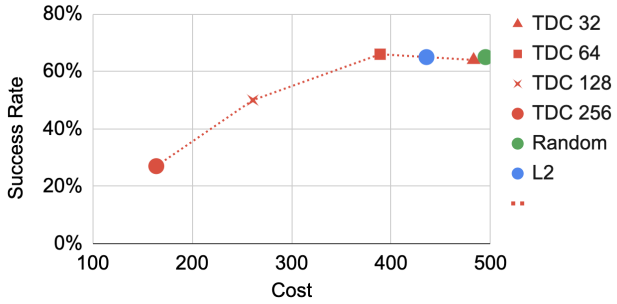

Figure 8: Comparison of methods for biasing the search with BELT and the task-model. TDC with different values for d_{cutoff}, a fully random search, and a L2 biased search are shown. These represent tradeoffs between exploiting low cost paths (low success and low cost) and exploring more (higher success rate and higher cost). $d_{\text{cutoff}} = 64$ (red square) was chosen herein.

5.4 BELT Convergence

Figure 9 shows the convergence of BELT as the number of samples increases, showing that at 2500 samples BELT is still solving more chains of tasks and may in the limit converge to a high solution rate. The problem and state space considered is quite complicated even with a good underlying policy, so this is not unexpected. Interestingly, of these still to be solved, almost all included button presses (99% of the problems without button pushes were solved from Table 2). This is because the button press is a transient task, so sampling a goal state with it explicitly pressed is much less likely than other tasks. This could be remedied by rebalancing the dataset or biasing the search towards these states.

The task-model sees a similar curve, but shifted downward. This is primarily due to errors in modeling two tasks: "rotate left" and button presses. The "rotate left" was due to the discontinuity between 2π and 0, and motivates better angle representations. The aforementioned transient issue for button presses is doubly important for the task-conditioned model, which does not recreate the full trajectory, but rather just the end state. This could be remedied through either a model that recreates the full trajectory or a model that also predicts task completion.

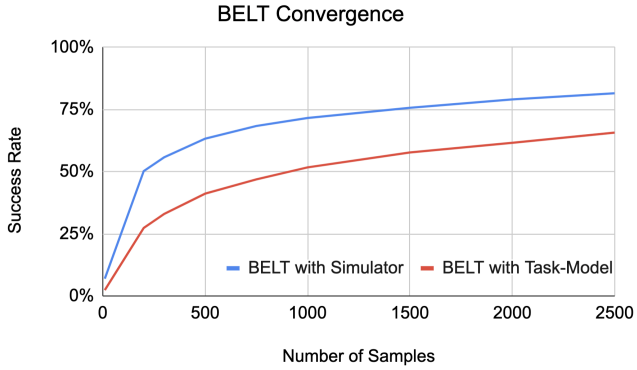

Figure 9: The convergence of BELT's success rate (with a simulator and the task-conditioned model) as the number of samples increases.

5.5 Evaluation on Only Goal Conditioned Problems

In section 3.2 we compared all methods across all chains, except for LMP, which can only be run when a single-goal satisfied all subtasks. This existed for 36% of the chains. In Table 3 we show the performance of the comparison methods on only these problems, which tend to be easier. The performance of all comparison methods increases on these problems.

Algorithm	Model	Solution Found	Success Rate	Feasible
LMP	–	–	6%	21%
CEM	Simulator	7%	38%	56%
CEM	Task-Model	11%	18%	35%
BELT	Action-Model	100%	5%	14%
BELT	Task-Model	84%	43%	83%
BELT	Simulator	99%	33%	75%

Table 3: Success rate and robustness for BELT, CEM, and LMP on only problems for which a single goal state satisfied the full chain (to evaluate LMP). Plan robustness is measured by the percent of solutions executed successfully (success rate) and if there was at least one successful in nine attempts (feasible).

5.6 Theoretical Results.

Briefly we note the lack of theoretical results for this work. These are precluded by the use of (1) a non-uniform sampling distribution and (2) a learned policy which both bias the search to useful regions, limiting full coverage of the state space. The learned model also can be inaccurate, limiting guarantees. While each of these is required to make BELT feasible in such a complex domain, without them the theoretical results from [10] and [29] can be trivially applied.

5.7 Metrics

In this section we formally define the comparison metrics and discuss what they mean as well as their interplay. The metrics considered are:

- *Solution Found* captures whether the planner returns that it has found a solution sequence of tasks for which the trajectory satisfies the success criteria. This metric primarily indicates an algorithm's ability to explore the space. Given N_{total} total chains and N_{solved} solutions returned, "solution found" is $N_{\text{solved}}/N_{\text{total}}$. This solution may have been fully played out in the simulator or imagined via the model, and thus this solution may not be robust or even feasible (as measured by the subsequent two metrics).
- *Success Rate* captures the robustness of a solution when rolled out. Given a solutions is returned and replayed N_{rollouts} times, of which $N_{\text{successful}}$ are successful, "success rate" is $N_{\text{successful}}/N_{\text{rollouts}}$. This may be robustness to a stochastic policy, to the environment, or to the learned model used for planning. Thus, this metric can be capped by the performance of the underlying policy.
- *Feasiblity* captures whether the solution returned is feasible after a number of rollouts. Given a solutions is returned and replayed N_{rollouts} times, if any of those N_{rollouts} is successful, the solution is called feasible. This metric captures the quality of the plan to be replayed even with an imperfect policy.

Note that each of these has some interplay, i.e., if solutions are rarely found then they are likely found for the easiest of problems only, so success rate and feasibility may be higher. We see this with CEM in particular.

Table 1 shows the performance of each algorithm against these three metrics. In terms of finding solutions, we find that CEM performs poorly and finds solutions at low rates. BELT performs well, finding solutions with the simulator 82% of the time, the action-conditioned model 100% of the time, and the task-conditioned model 66% of the time. This indicates that BELT is able to explore the state space effectively. Much of the reduced solution rate of BELT with the task-conditioned model stems from "rotate left" tasks, for which the task-model has difficulty modeling the discontinuity from 2π to 0.

In terms of success rate, i.e., robustness in replaying the solution, Chain-LMP (an oracle sequence of tasks) is able to replay 32% of the time, demonstrating both the difficulty of each subtask in this environment and limitations of the policy. LMP is not able to perform the long-horizon tasks effectively and is successful in only 6%. CEM with the simulator is relatively robust on the few problems it was able to solve, while performance with the CEM task-model is poor. BELT with the action-conditioned model was able to find a solution for each task, but the performance is very poor, indicating the action-conditioned model being recursively called became unstable. In this instability it can output erroneous states that may confuse the success detectors, thus returning poor solutions. BELT with the task-conditioned model and the simulator is able to solve nearly at the rate of the oracle (27% and 26%, respectively).

In terms of feasibility, with a perfect sequence of subtasks, the oracle Chain-LMP performs well at 87%. For these values, each plan was replayed 9 times and we expect that with more runs this should converge to nearly 100%. The plans that leverage the simulator achieve 58% (CEM) and 68% (BELT), despite using the simulator. While these too should increase with more rollouts, this may indicate that the planners sometimes exploit unrepeatable errors in the simulator, which can particularly be an issue with Mujoco. Between the task-model and the simulator for BELT, the primary loss of feasibility is in chains including button pushes, which can be difficult to model due to its transient nature and required precision.

5.8 Application to Dynamic and Unknown Environments.

In settings where the environment is dynamic or not fully observed, BELT plans may be limited in horizon. In these settings, the BELT tree can be reused to replan, either by rewiring based on changes or by planning trajectories back into the tree in case of plan divergence. Furthermore, concepts from BELT can be used to form a graph and maintain a representation of possible paths through the environment.

5.9 Full-Page Figures

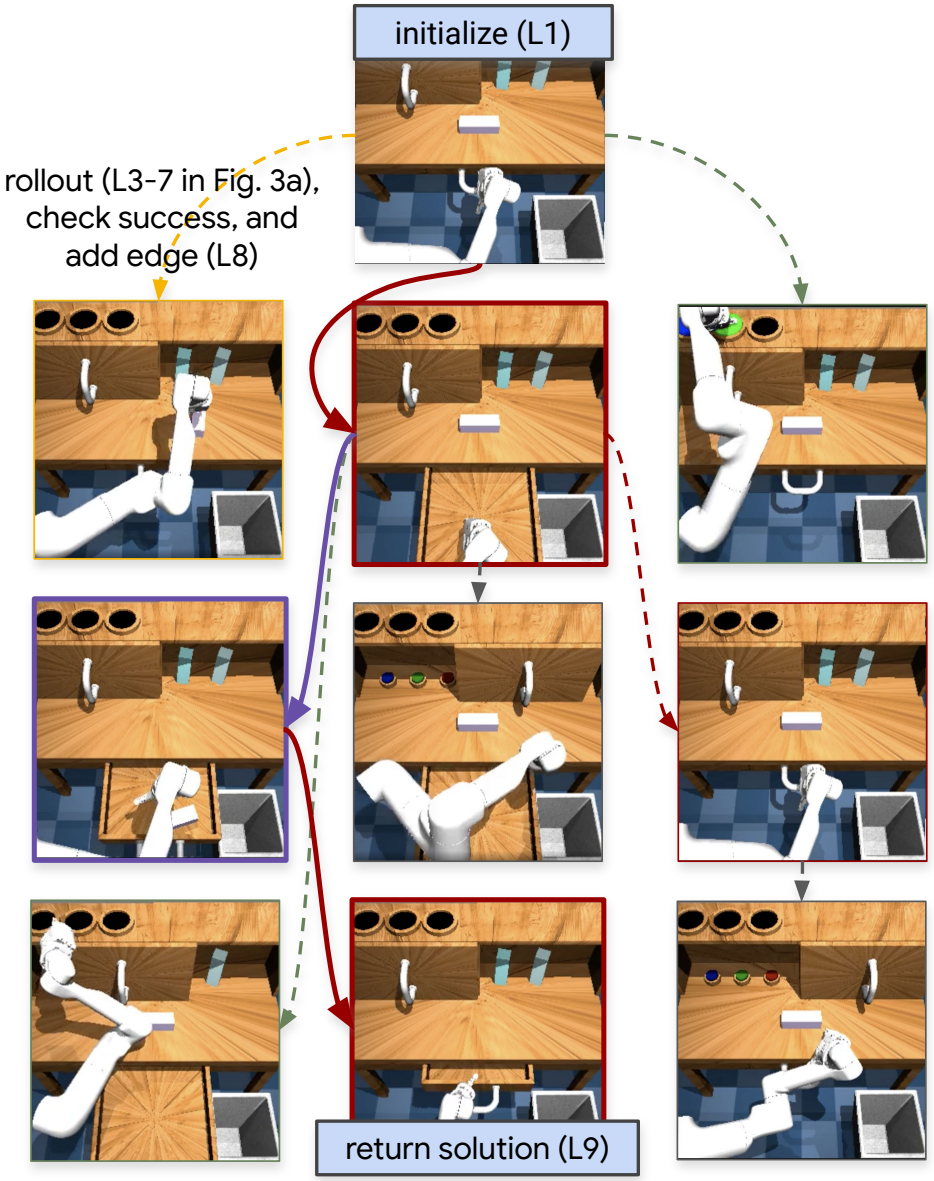

Figure 10: Full-page version of Fig. 3b. At each iteration BELT samples a goal from demonstration data and a tree state and rolls out a task-conditioned model to add edges to the tree. The solution trajectory is selected as the lowest cost solution, as verified by the binary success check `Success`.

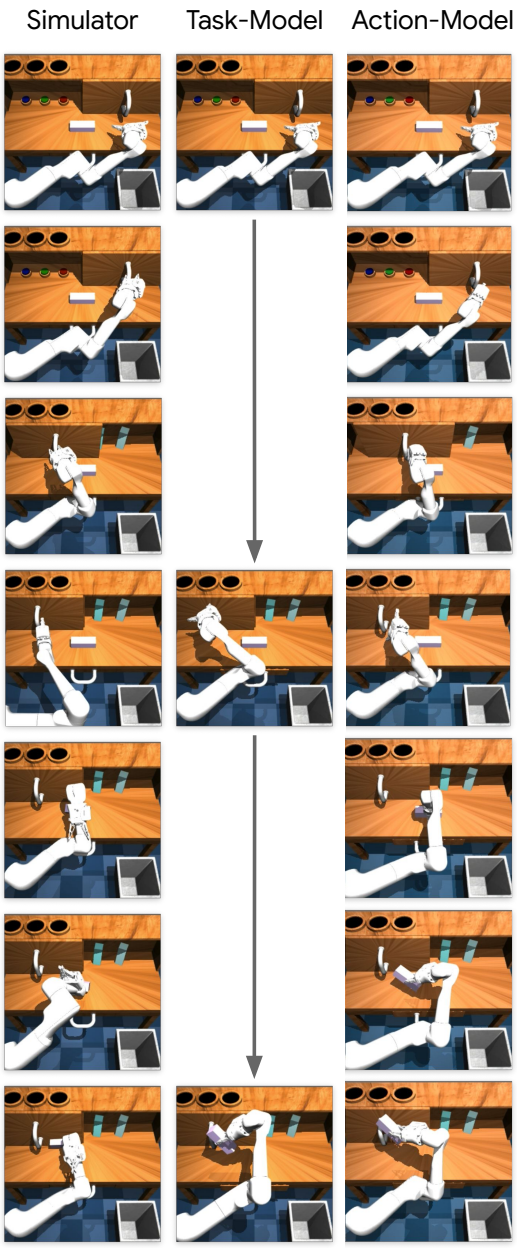

Figure 11: Full-page version of Fig. 4c. Task-models, action-model, and simulator our experiment environment. (video link).

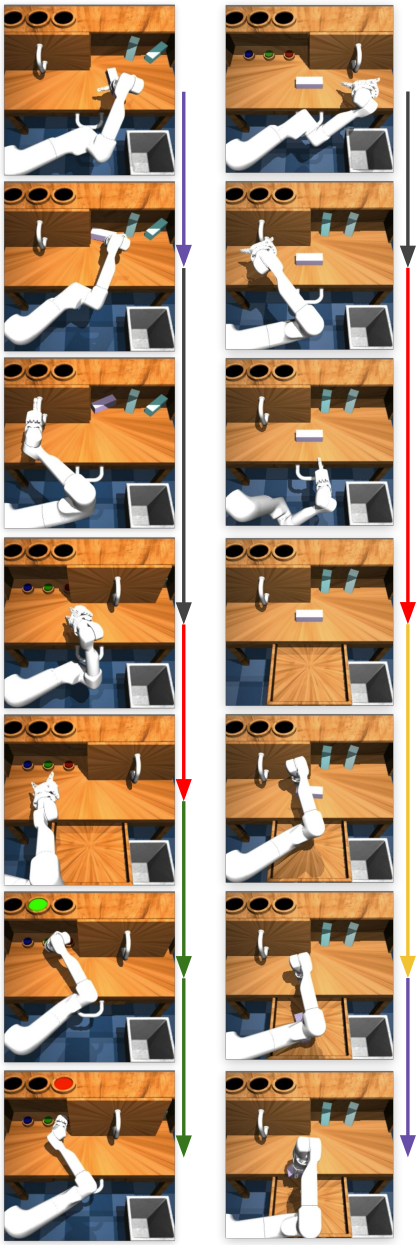

Figure 12: Full-page version of Fig. 5. Plans from BELT, demonstrating its ability to plan long-horizon, sequential trajectories (video link).