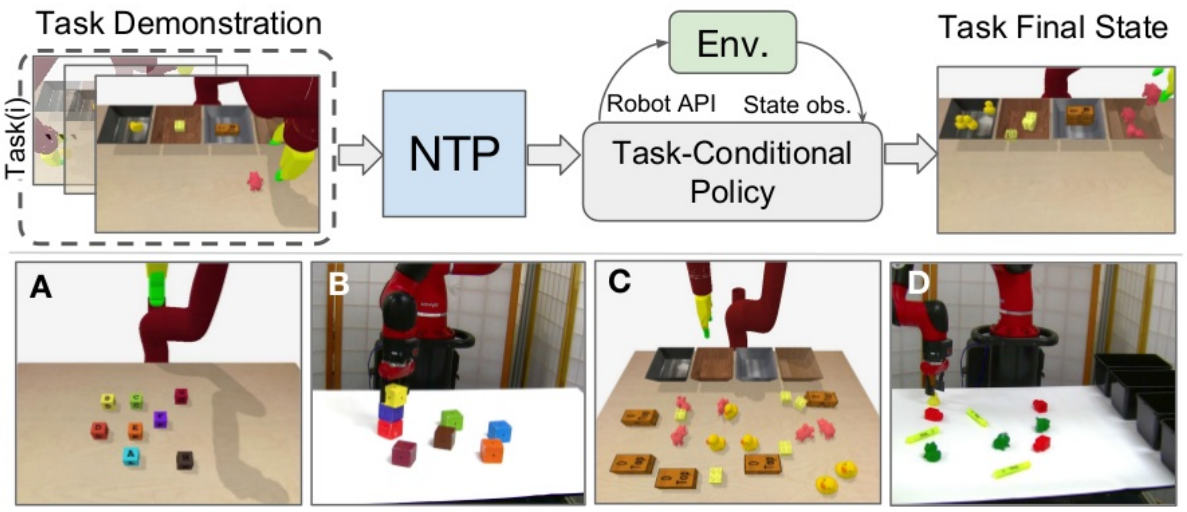
1710.01813 - Neural Task Programming: Learning to Generalize Across Hierarchical Tasks

- **Yunqiu Xu**
- Other reference:
  - http://technews.cn/2017/10/12/neural-task-programming-learning-to-generalize-across-hierarchical-tasks/
  - https://www.youtube.com/watch?v=THq7I7C5rkk&feature=youtu.be
  - https://www.youtube.com/watch?v=Lcxz6dtYjI4

---

1. Introduction

- Challenge:
  - Learning policies that generalize to new task objectives $\rightarrow$ meta learning
  - Hierarchical composition of primitives for long term environment interactions $\rightarrow$ HRL
- Our work: NTP, connect few-shot learning from demonstration with neural program induction
- Input: task specification, what is it?
  - A time-series that describes the procedure and the final objective of a task
  - Can be task demonstration or a list of language instructions
  - **In this work we use task demonstration**
    - Location trajectories of objects
    - 3rd person video demonstration



- Procedure:
    - Left: NTP interprets a task specification
    - Middle: Instantiate a hierarchical policy as neural program
    - Right: Decode the objective from input specification into subtasks
    - Hierarchical policy will interact with the env until the goal is achieved
    - Buttom: primitive actions to interact with env directly
- Task variation:
    - Length: number of steps (e.g. having more objects to transport)
    - Topology: permutations and combinations of sub-tasks (e.g. manipulating objects in different orders)
    - Semantics: task definitions and success conditions (e.g placing objects into a different container)
- Contribution:
    - NTP: enable meta-learning on hierarchical tasks
    - NTP on single-arm robot tasks: Block Stacking, Object Sorting, and Table Clean-up
    - Enables knowledge transfer and one-shot demonstration
    - NTP can can be trained with visual input end-to-end

2. Background & Related Work

- Skill learning:
  - Goal : Learn policies that adapt to new task objectives
  - Model-based approach:
    - Segment into hand-engineered state machine
    - Challenge in scaling
  - Learning-based approach:
    - Do not need manually designed state representation
    - Need task-specific reward functions
- Learning from Demonstrations: **imitation**
  - Do not need to define state machines / reward functions
  - Goal: learn policies which generalize beyond provided examples
  - BC is an example
  - CHallenges: still not robust enough to generalize
- Few-Shot Generalization in LfD: **Our work**
  - Similar work: **MIL / one-shot imitation learning**
  - Goal : Learns to learn from an input task specification during training
  - Testing: generates a policy conditioned on a single demonstration provided as a time-series showing the task execution
  - 现存方法没法处理long term / sparse reward的问题, 因此我们的工作结合HRL $\rightarrow$ 可以与MLSH对比下
- Hierarchical Skill Composition
  - Goal : Enable long-term robot-environment interaction
  - Traditional method: option / subtask framework, **对任务敏感, 没法保证在不同任务目标上的复用性**
  - **因此我们将HRL与meta-learning结合, 在保证复用性的同时尽可能解决复杂任务**
- Neural program induction:
  - Goal : Learn a latent program representation that generates program outputs
  - E.G. NPI
    - Generalize on task length
    - Use a task-agnostic recurrent neural network to
      represent and execute programs
    - Trained with richer supervision from the full program execution traces
    - High data efficiency

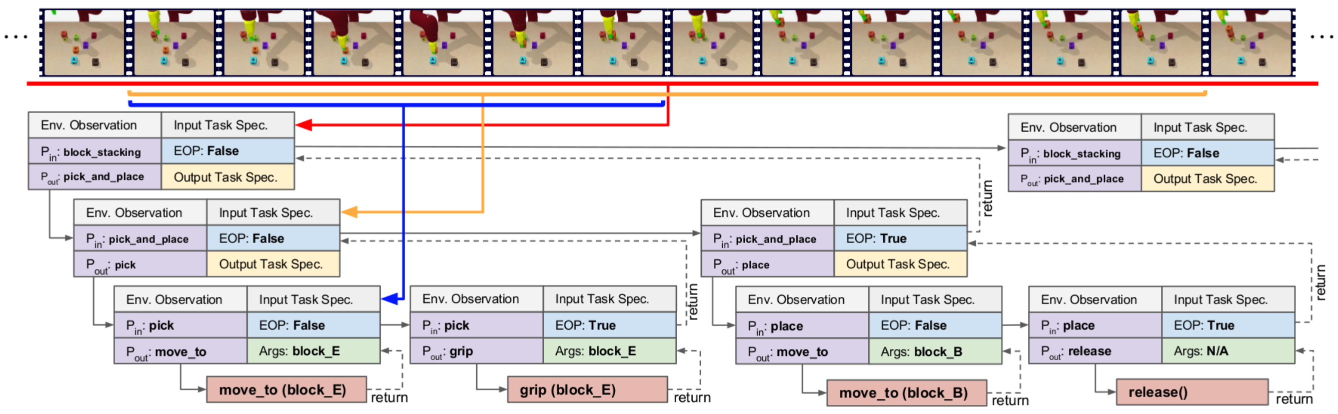
    - Challenge: hard to generalize without training
- NTP (our work):
    - Learns to instantiate neural programs given demonstrations of tasks, then generalizing to unseen tasks/programs
    - Decomposes the overall objective into simpler objectives recursively
    - For each subtask, NTP delegates a neural program to perform this task
    - Previous work: execute a pre-defined task one at a time
    - Our work: more general genralization
        - Longer lengths as NPI
        - Sub-task permutations (topology) and success conditions (semantics)

3. Problem Formulation

- Environment and tasks:
    - T : the set of all tasks
    - S : the env state space
    - O : the observation space
    - For each task $t \in T$, success condition is $g : S \times T \rightarrow \{0, 1\}$
        - If $g(s, t) = 1 \rightarrow$ this task is completed in state s at time t
- Task specification: $\Psi(t) = \{x_1, x_2, \ldots, x_N\}$
    - This is a time series that describes the procedure and the final objective of the task
    - Form 1: trajectories of object locations
    - Form 2: raw video sequences
- Our goal:
    - Given a task specification, learn a "meta-policy" $\hat{\pi} : \Psi \rightarrow (O \rightarrow A)$
    - When test, given the specification of a new task $\Psi(t)$, $\hat{\pi}$ generates to $\pi(a|o; \Psi(t)) : O \rightarrow A$
    - By using this generated task, the agent can reach the termination state s_T that $g(s_T, t) = 1$

3.1 Why use Neural Programming for LfD

- Previous LfD method (e.g. Duan et al, 2017. One-Shot Imitation Learning): goal-driven policy, can not exploit compositional task structures to modularization
- NTP:
    - Extend the architecture of NPI $\rightarrow$ **not only problem size, but also topology and semantics**
    - Core network:
        - Task-agnostic, decide which sub-program to run next
        - Feeds a subset of task specification to next program
    - Recursively decomposes a task specification and solves a hierarchical task by divide-and-conquer



- Block stacking example
    - 任务描述: 将箱子按编号垒起来, e.g. "D"在"E"上面
    - Top level program "block stacking"
        - 这个是总任务, 输入整个(红色区域) demonstration
        - 输出sub-program "pick and place"
    - Sub-program "pick and place"
        - 输入橙色区域demonstration
        - 输出sub-program "pick"
    - Sub-program "pick"
        - 输入蓝色区域demonstration
        - 输出并调用API "move to"
    - EOP:
        - End of program
        - If true, go back to its caller program
        - 例如第三排第二个sub-program EOP为true, 返回到第二排第一个sub-program,

然后前往第二排第二个sub-program

3.2 Neural Programmer-Interpreter

- 之前提到NTP是NPI的延伸, 这里对NPI做下介绍: Reed & Freitas Neural Programmer-Interpreters
- NPI is a type of neural program induction algorithm
    - Goal of network: imitate the behavior of a computer program
    - Invoke programs recursively given certain context
    - Stop the current program and return to upper-level programs
- Controller:
    - Input:
        - State s_i
        - p_i : Program embedding retrieved from learnable key-value memory structure $[M^{key}, M^{prog}]$
        - a_i : Current arguments, 注意这个和RL不同不是action
    - Output:
        - A program key to invoke sub-program p_{i+1}
        - Argument to next program a_{i+1}
        - EOP probability r_i
- Architecture
    - A domain-specific encoder: $s_i = f_{enc}(o_i, a_i)$
    - **Core module: LSTM**
        - At time t_i, given o_i and h_{i-1}, it selects the next program to run
        - $h_i = f_{lstm}(s_i, p_i, h_{i-1})$
    - An output decoder: $r_i, p_{i+1}, a_{i+1} = f_{dec}(h_i)$
- How NPI be executed

    - LSTM core is shared across all tasks
    - If r_i exceeds threshold $\rightarrow$ end this sub program and go back to the called program
    - If the program is not primitive $\rightarrow$ call a sub-program as well as its arguments
    - If the program is primitive $\rightarrow$ execute low-level actions to interact with environment

- Pros and Cons:
    - High data efficiency
    - Can only vary problem size
    - Our work will extend it to enable task topology and semantics

4. Neural Task Programming

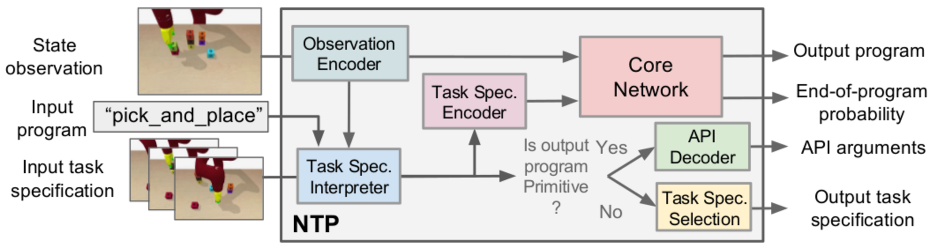
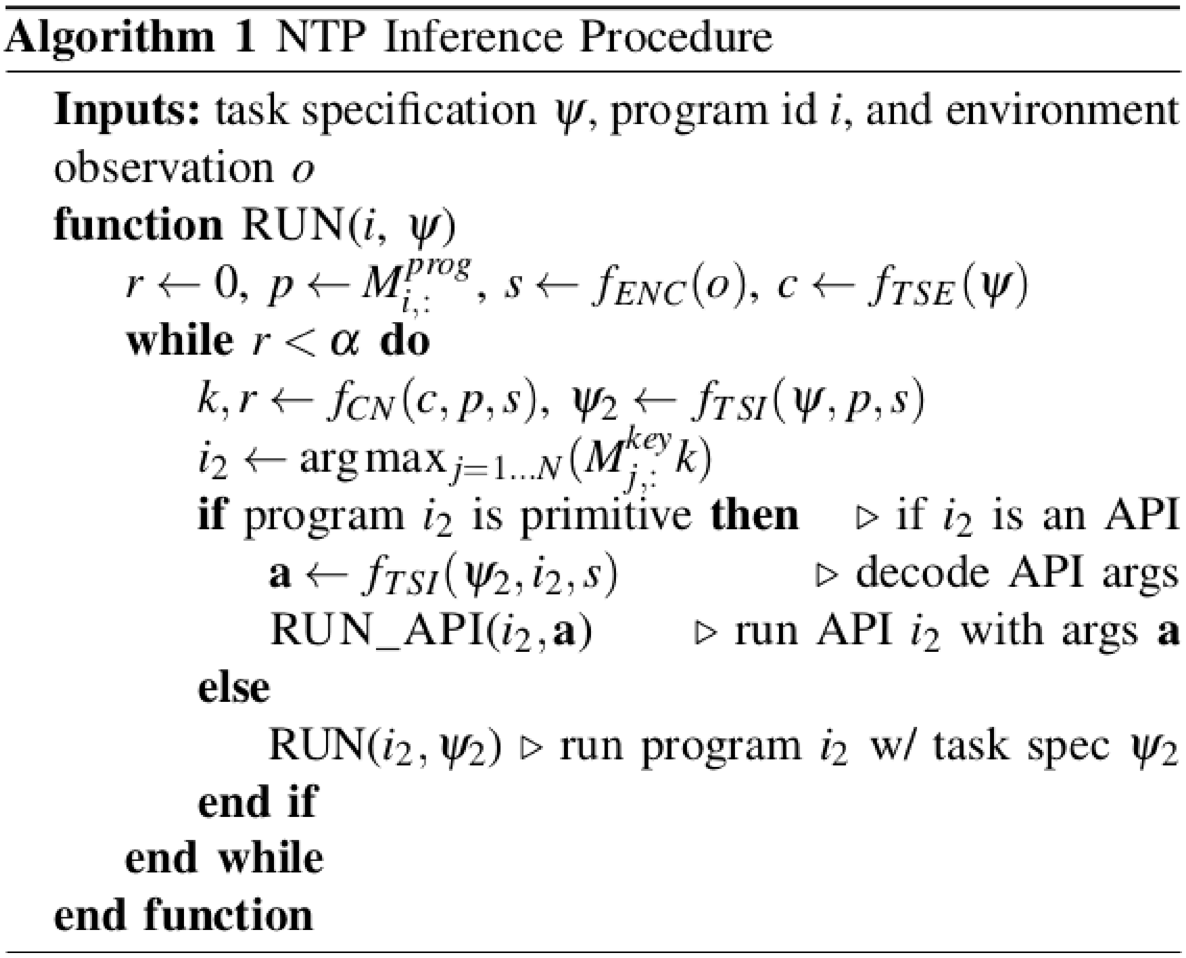


Fig. 2: **Neural Task Programming (NTP)**: Given an input program, a task specification, and the current environment observation, a NTP model predicts the sub-level program to run, the subsequence of the task specification that the sub-level program should take as input, and if the current program should stop.

- Architecture:
    - f_{TSE} : task specification interpreter
        - Input : s_i, p_i, Ψ_i
        - Transform a task specification Ψ_i into vector space
        - If current program p is not primitive $\rightarrow$ predict sub-specification for next sub-program
        - If current program p is primitive $\rightarrow$ predict arguments for robot APIs
        - **Scoping constraint: Ψ_{i+1} is a constiguous subsequence of Ψ_i**
    - f_{CN} : core network
        - Input : s_i, p_i, Ψ_i
        - Output : next sub-program to invoke p_{t+1}, EOP probability r_i
    - f_{TSI} : task specification interpreter
    - Note that NTP also has key-value memory to store and retrieve embeddings:
        - Given predicted key k_i, $j^* = argmax_{j=1...N}(M_j^{key} : k_i)$
        - Output $p_i = M_{j^*}^{prog}$
- Difference from NPI:
    - NTP can interpret **task specifications** and perform **hierarchical decomposition** $\rightarrow$ can be considered as a meta-policy
    - Uses **APIs** as the buttom-level actions $\rightarrow$ scale up neural programs for

complex tasks
  - Reactive core network instead of a recurrent network $\rightarrow$ less history-dependent
- Algorithm:



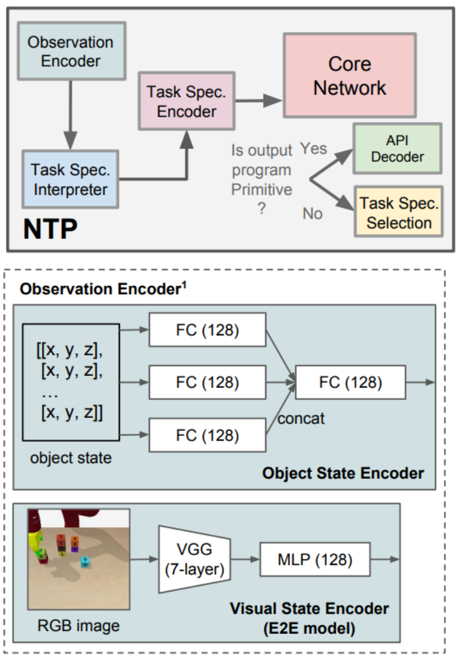
Algorithm 1 NTP Inference Procedure

```
Inputs: task specification $\psi$, program id $i$, and environment observation $o$
function RUN($i$, $\psi$)
    $r \leftarrow 0$, $p \leftarrow M^{prog}_{i,:}$, $s \leftarrow f_{ENC}(o)$, $c \leftarrow f_{TSE}(\psi)$
    while $r < \alpha$ do
        $k, r \leftarrow f_{CN}(c, p, s)$, $\psi_2 \leftarrow f_{TSI}(\psi, p, s)$
        $i_2 \leftarrow \arg\max_{j=1...N}(M^{key}_{j,:} k)$
        if program $i_2$ is primitive then        ▷ if $i_2$ is an API
            $\mathbf{a} \leftarrow f_{TSI}(\psi_2, i_2, s)$        ▷ decode API args
            RUN_API($i_2$, $\mathbf{a}$)        ▷ run API $i_2$ with args $\mathbf{a}$
        else
            RUN($i_2$, $\psi_2$) ▷ run program $i_2$ w/ task spec $\psi_2$
        end if
    end while
end function
```

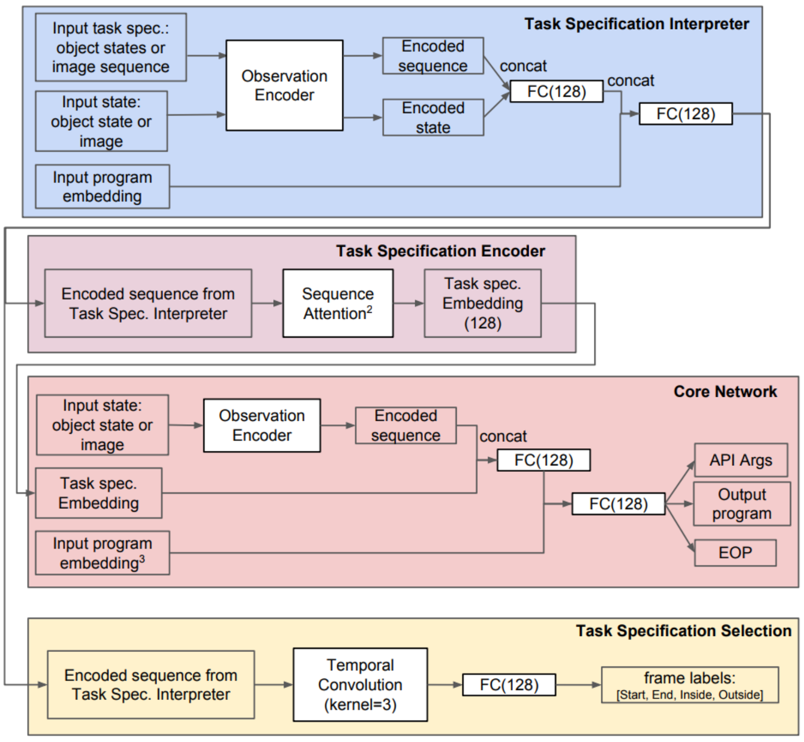
- How to choose subsequence: CNN architecture
  - Input: $\mathbf{\Psi}_i$
  - Output: softmax probability of scoping labels $Pr_j(l \in \{start, end, inside, outside\})$
  - Scoping labels: whether the location is start/end or inside/outside the subsequence
  - Then we use these probabilities to decode the optimal subsequence as the output sub-task specification $\mathbf{\Psi}_{i+1}$
  - Decoding: taking the start and end points with the highest probabilities (**理论上应该用最长连续子序列, 但本文用这个效果更好**)
- How to train NTP:

- Use rich supervision from program execution traces
- Each execution trace is a list of tuples $\{\xi_t|\xi_t = (\Psi_t, p_t, s_t), t = 1...T\}$
- Goal: maximize the probability of correct executions over all tasks, $\theta^* = \sum_D logPr[\xi_{t+1}|\xi_t; \theta]$

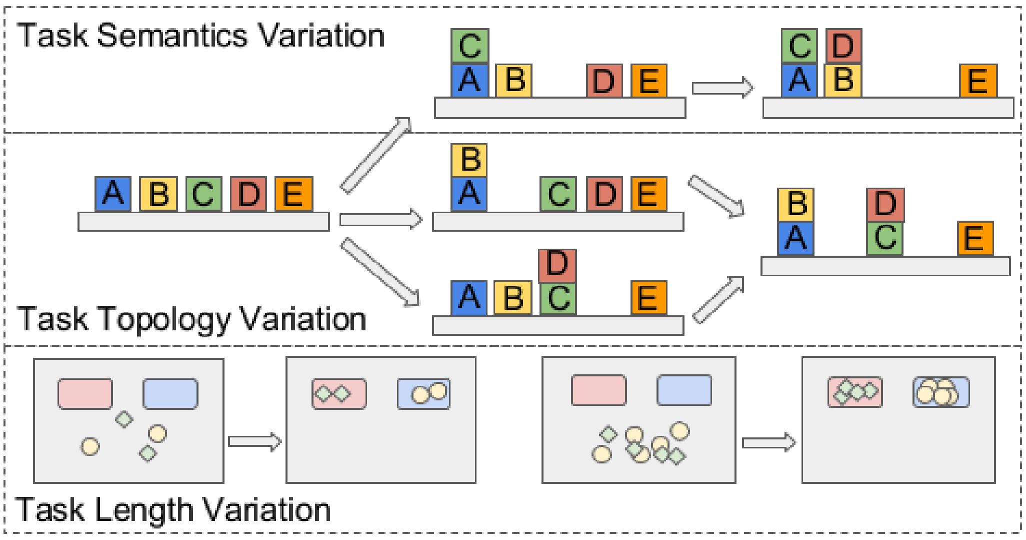
- What does an expert policy / demonstration look like
  - An agent with hard-coded rules: calls programs to perform a task
  - The state representation in the task demonstrations is in the form of object position trajectories relative to the gripper frame
- Implementation details:



1. The weights of the observation encoder is shared across the network.
2. Weighted sum of the encoded sequence.
3. Program embeddings are randomly initialized.


Task Specification Interpreter
Input task spec.: object states or image sequence
Input state: object state or image
Input program embedding
Observation Encoder
Encoded sequence
Encoded state
concat
FC(128)
concat
FC(128)
Task Specification Encoder
Encoded sequence from Task Spec. Interpreter
Sequence Attention[2]
Task spec. Embedding (128)
Core Network
Input state: object state or image
Observation Encoder
Encoded sequence
concat
FC(128)
Task spec. Embedding
Input program embedding[3]
FC(128)
API Args
Output program
EOP
Task Specification Selection
Encoded sequence from Task Spec. Interpreter
Temporal Convolution (kernel=3)
FC(128)
frame labels: [Start, End, Inside, Outside]

- Task variations:



5. Experiment

- Questions:
  - Does NTP generalize to changes in all three variations
  - Can NTP use image input without access to ground truth state
  - Can NTP tackle real-world tasks (combination of three variations)
- Tasks to test:
  - Object Sorting
  - Block Stacking
  - Table Clean-up
- Baselines:
  - Flat: similar to one-shot imitation learning, non-hierarchical model
  - Flat(GRM): Flat with GRU cell
  - NTP(no scope): feeds the entire demonstration to the subprograms, no scoping constraint (可以看下之前对scoping constraint的定义)
  - NTP(GRU): NTP model with GRU cell

5.1 Object Sorting

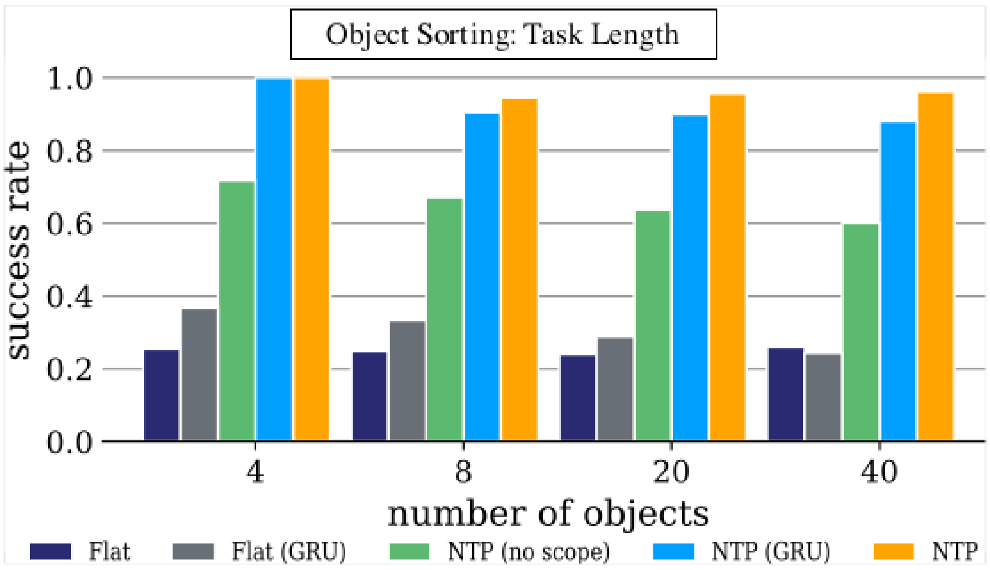


Fig. 5: **Task Length**: Evaluation of the Object Sorting in simulation. The axes represent mean success rate (*y*) with 100 evaluations each and the number of objects in unseen task instances (*x*). NTP generalizes to increasingly longer tasks while baselines do not.

5.2 Block Stacking

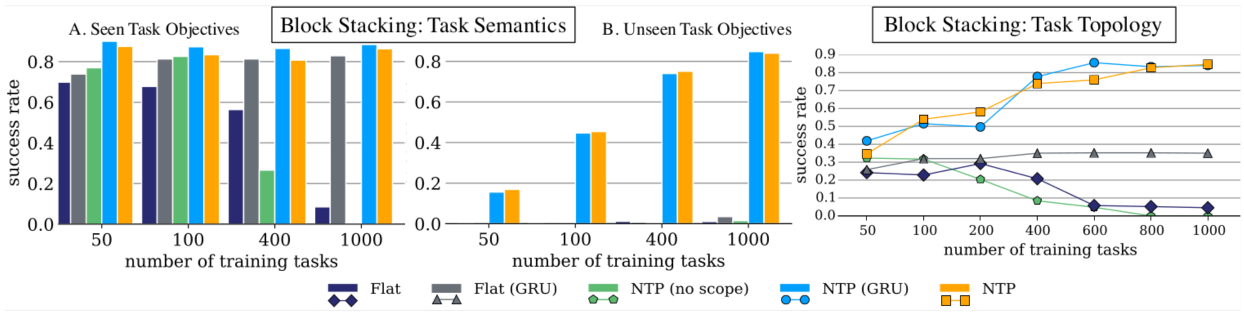


Fig. 6: **Task Semantics**: Simulated evaluation of the Block Stacking. The *x*-axis is the number of tasks used for training. The *y*-axis is the overall success rate. (**A**) and (**B**) show that NTP and its variants generalize better to novel task demonstrations and objectives as the number of training tasks increases.
Task Topology: Simulated evaluation of the Block Stacking. NTP shows better performance in task topology generalization as the number of training tasks grows. In contrast, the flat baselines cannot handle topology variability.

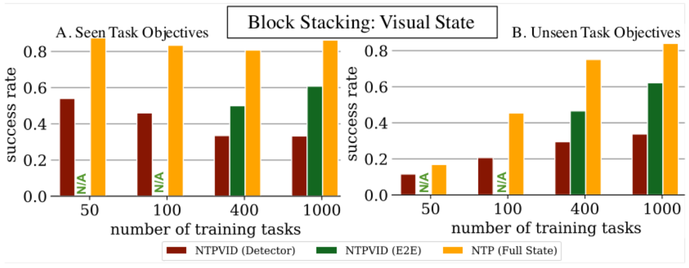


Fig. 7: **NTP with Visual State:** NTPVID (Detector) uses an object detector on images which is subsequently used as state in NTP. NTP (E2E) is an end to end model trained completely on images with no low-level state information. We note that in the partial observation case (only video), similar learning trends were observed as compared to fully observed case (NTP (Full State)), albeit with a decrease in performance.

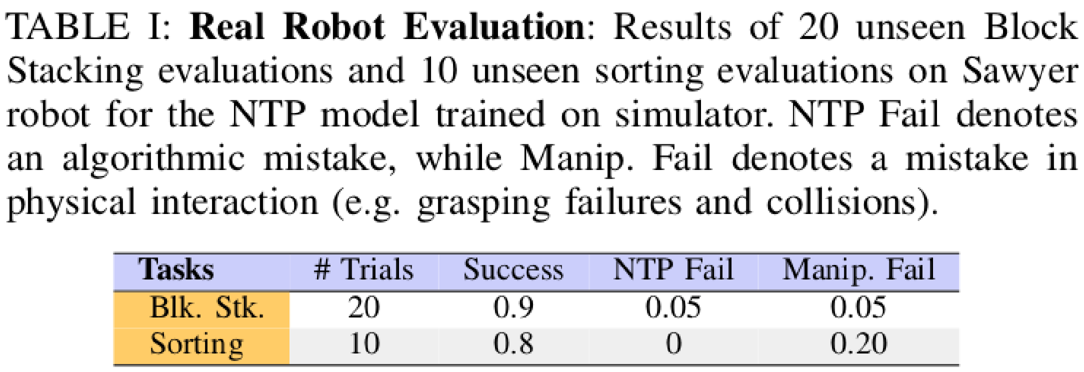


TABLE I: **Real Robot Evaluation**: Results of 20 unseen Block Stacking evaluations and 10 unseen sorting evaluations on Sawyer robot for the NTP model trained on simulator. NTP Fail denotes an algorithmic mistake, while Manip. Fail denotes a mistake in physical interaction (e.g. grasping failures and collisions).

Tasks	# Trials	Success	NTP Fail	Manip. Fail
Blk. Stk.	20	0.9	0.05	0.05
Sorting	10	0.8	0	0.20

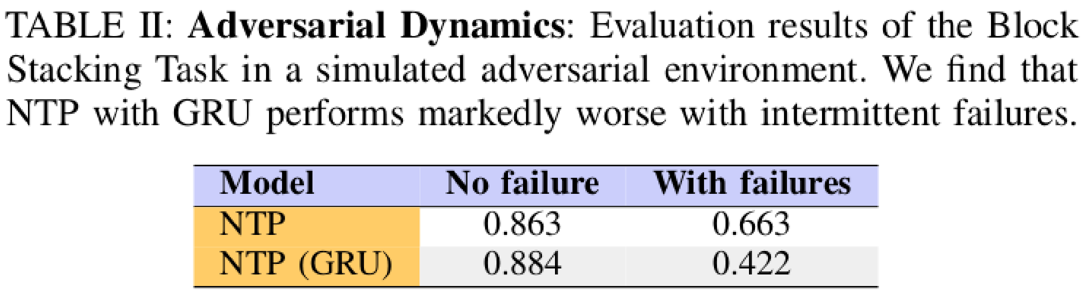
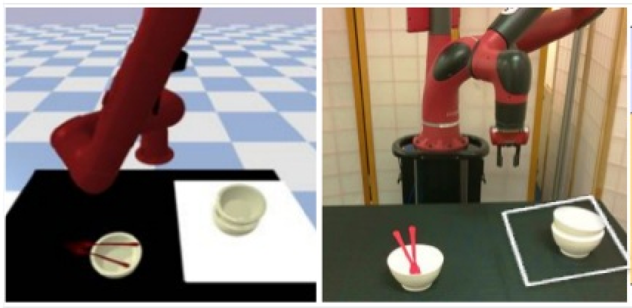


TABLE II: **Adversarial Dynamics**: Evaluation results of the Block Stacking Task in a simulated adversarial environment. We find that NTP with GRU performs markedly worse with intermittent failures.

Model	**No failure**	**With failures**
NTP	0.863	0.663
NTP (GRU)	0.884	0.422

5.3 Table Clean-up



# Bowls, # Forks	Success
2 B, 1 F	1.00
2 B, 2 F	0.95
3 B, 2 F	0.75
3 B, 3 F	0.55

Fig. 8: **Table Clean-up**: in simulated and real environment.The table shows success rates for varying numbers of forks and bowls in simulated evaluation.

6. Summary

- NTP:
  - Similar to MLSH, combine meta-learning with HRL
  - Compared with goal-driven learning, NTP exploits compositional task structures to modularization
  - Extend NPI to NTP: more variations
- 和之前看的MLSH相比, 不需要考虑瞎初始化的问题, 但本文没给源代码, 有些细节还是不大理解, 感觉不好实现