Skill Transfer for Temporally-Extended Task Specifications: Supplementary Material

Jason Xinyu Liu[*], **Ankit Shah**[*], **Eric Rosen, George Konidaris, Stefanie Tellex**
Department of Computer Science
Brown University
{xinyu_liu, ankit_j_shah, eric_rosen, george_konidaris, stefanie_tellex}
@brown.edu

1 Appendix

1.1 Edge matching criteria

LTL-Transfer uses a set of edge matching conditions to prune the infeasible edges from the reward machine (RM) graph given a set of transition-centric options and to determine the candidate options given the current RM state and a particular outgoing edge to target. Below, we present two edge matching conditions, *constrained* and *relaxed*, that both ensure progression of the reward machine towards and the transition leading to non-failure states.

Without the loss of generality, we assume the test specification is φ_{test}, the test task is in RM state q, where the self-transition edge is $e^{\varphi_{test}}_{q,q}$ and the targeted edge transition is $e^{\varphi_{test}}_{q,q'}$. We are determining the applicability of the transition-centric option o_{e_1,e_2}.

1.1.1 Constrained edge matching criterion

The *constrained* edge matching criterion ensures every truth assignment that satisfies the self transition edge $e^{\varphi_{test}}_{q,q}$ of RM also satisfies the self transition edge e_1 of the transition-centric option, and the same condition holds for the outgoing edge $e^{\varphi_{test}}_{q,q'}$ of RM and the outgoing edge e_2 of the option. Mathematically, the following equation must evaluate to true to declare a constrained match, where asg and asg' represent truth assignments, $sat(f, asg)$ is a function that evaluates the Boolean formula f on the truth assignment asg, and $sat\_models(f)$ returns all truth assignments that satisfy the Boolean formula f. We used the Sympy library for logic operations [2].

$$sat(e_1, asg) \ \wedge sat(e_2, asg') \ \forall asg \in sat\_models(e^{\varphi_{test}}_{q,q}), \ \forall asg' \in sat\_models(e^{\varphi_{test}}_{q,q'}) \quad (1)$$

1.1.2 Relaxed edge matching criterion

The *relaxed* edge matching criterion ensures that the self transition edges e_1 and $e^{\varphi_{test}}_{q,q}$ share satisfying truth assignments, so are the outgoing edges e_2 and $e^{\varphi_{test}}_{q,q'}$. We let the failure edge $e^{\perp}$ be the edge from the current RM state q to the failure state $q^{\perp}$ if one exists. To prevent the selection of a transition-centric option o_{e_1,e_2} that violates constraints, we enforce that both self and outgoing edges of the option, e_1 and e_2, must not share any satisfying truth assignment with the failure edge $e^{\perp}$ if one exists. Lastly, to guarantee RM progression after applying the transition-centric option, the *relaxed* edge matching condition ensures that the outgoing edge e_2 of the option must share no satisfying truth assignments with the self transition edge $e^{\varphi_{test}}_{q,q}$ of the RM. Mathematically, the following equation must evaluate to true to declare a relaxed match, where $sat(f)$ is a function that determines if any truth assignment exists to satisfy the Boolean formula f.

$$sat(e_1 \wedge e^{\varphi_{test}}_{q,q}) \wedge sat(e_2 \wedge e^{\varphi_{test}}_{q,q'}) \wedge \neg sat(e_1 \wedge e^{\perp}) \wedge \neg sat(e_2 \wedge e^{\perp}) \wedge \neg sat(e_2 \wedge e^{\varphi_{test}}_{q,q}) \quad (2)$$

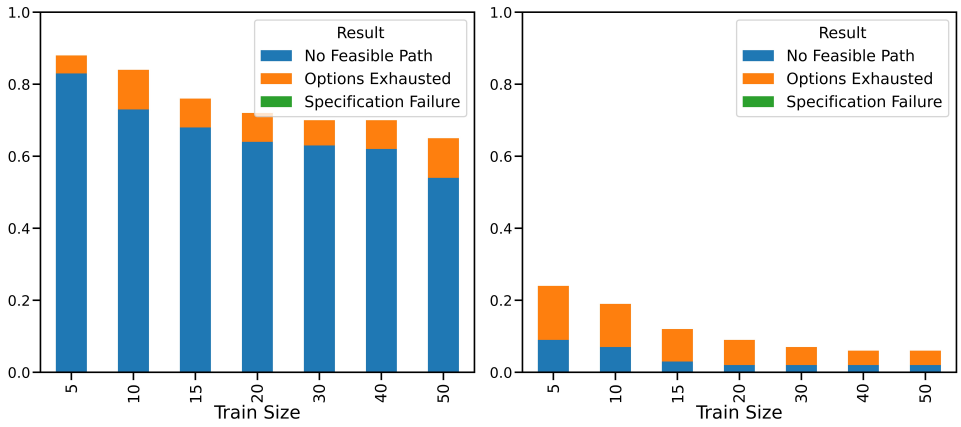
[*]Equal contribution.

Preprint. Under review.

1.2 Full Results

Cause of failure: As described in our draft, we logged the reason for failure for each unsuccessful transfer attempt as one of three possible causes: *specification failure*, where the agent violates a constraint and the reward machine is progressed to an unrecoverable state; *no feasible path*, where there are no matched transition-centric options for paths connecting the start state to an accepting state; *options exhausted*, where there are no further transition-centric options available to the agent to further progress the state of the task.

Figure 1 depicts the relative frequency of the failure modes when the agent is trained and tested on *mixed* task specifications. Note that with the *Constrained* edge-matching criterion, absence of feasible paths connecting the start and the accepting state is the primary reason for failure (Figure 1a, whereas with the *Relaxed* edge-matching criterion, the agent utilizing all available safe options without progressing the task is the primary reason for failure (Figure 1b).



(a) Constrained (b) Relaxed

Figure 1: Reasons for failed task executions for agents trained and evaluated on *Mixed* task specification datasets. Note that all values are depicted in fractions.

Learning curve for various training datasets: Next, we present the results for learning curve of the success rate when transferring policies learned on different specification types.

The learning curves for training on formulas from the *Hard* training set with both the edge matching criteria are depicted in Figure 2.

The learning curves for training on formulas from the *Soft* training set with both the edge matching criteria are depicted in Figure 3.

The learning curves for training on formulas from the *Strictly Soft* training set with both the edge matching criteria are depicted in Figure 4.

The learning curves for training on formulas from the *No Orders* training set are being generated at the time of submission, and are expected to share nearly identical trends as the learning curves from the other training sets given the completed data points. We will include the plots in the final version of the paper.

Note that for training on each of the specification types, the learning curve trends are nearly identical to the learning curves on training with *Mixed* specification types as depicted in Figure 3 in the main draft. *Hard* specification types remain the most challenging specification ordering types to transfer to.

1.3 Selected Solution Trajectories

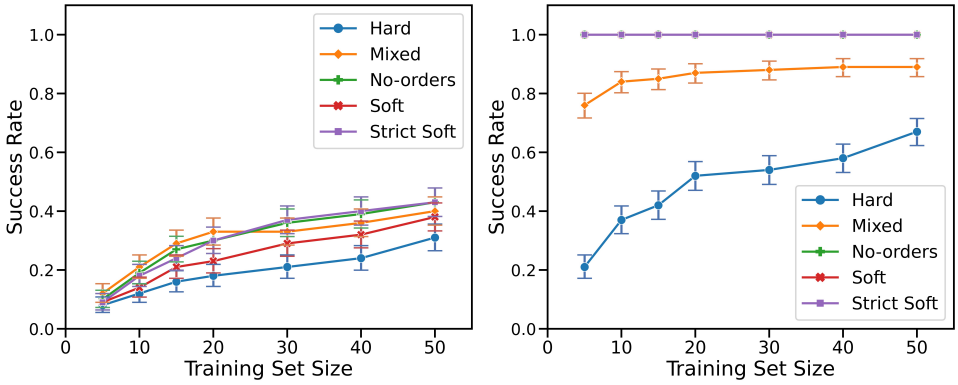
Consider the case with *mixed* training set with 5 formulas on *map 0*. The training formulas are:

- $\mathbf{F}grass \wedge \mathbf{F}shelter \wedge \mathbf{F}(wood \wedge \mathbf{XF}workbench)$
- $\mathbf{F}toolshed \wedge \mathbf{F}workbench \wedge \mathbf{F}shelter \wedge (\neg toolshed\ \mathbf{U}\ shelter) \wedge \mathbf{F}(grass \wedge \mathbf{F}bridge)$

- $\mathbf{F}toolshed \wedge \mathbf{F}(shelter \wedge \mathbf{F}(axe \wedge \mathbf{F}wood))$
- $\mathbf{F}iron \wedge \mathbf{F}(shelter \wedge \mathbf{XF}(bridge \wedge \mathbf{XF}factory))$
- $\mathbf{F}factory$

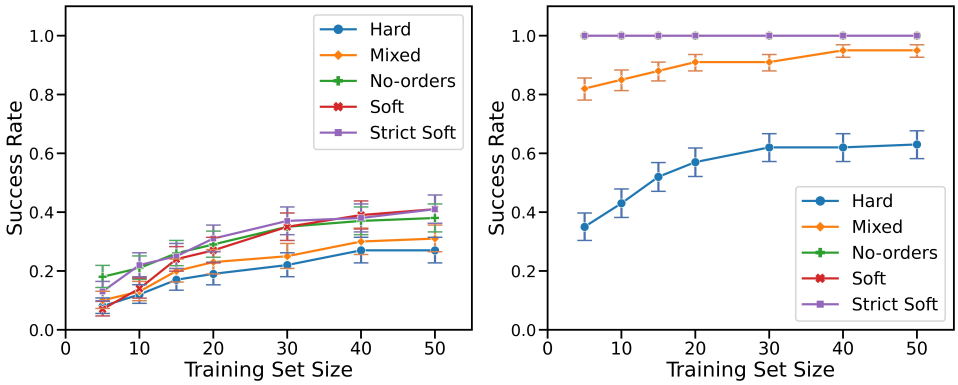
One of the *Mixed* test formulas was $\varphi_{test} = \mathbf{F}workbench \wedge \mathbf{F}grass \wedge \mathbf{F}axe$. The reward machine for this task specification is depicted in Figure 5a. Given the training set of formulas, and the use of the *Constrained* edge matching criterion, the start state is disconnected from all downstream states as no transition-centric options match with the edge transitions. Therefore, the agent does not attempt to solve the task and returns failure with the reason being *no feasible path*, i.e., a disconnected reward machine graph after removing infeasible edges.

If the *Relaxed* edge matching criterion is used, there are matching transition-centric options for each of the RM edges. The trajectory adopted by the agent when transferring the policies is depicted in Figure 6. The agent collects all the three requisite resources before it terminates the task execution. Further note that the agent passes through a $wood$ resource grid as the specification does not explicitly prohibit it.



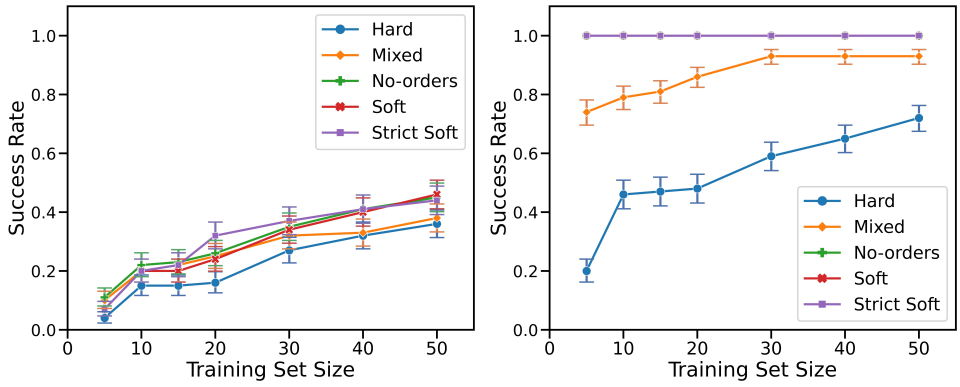
(a) Constrained (b) Relaxed

Figure 2: Figure 2a depicts the success rate of the agent trained on *Hard* training sets of various sizes using LTL-Transfer with the *constrained* edge-matching criterion when transferring to test sets of various specifications types. Figure 2b depicts the success rates with the *relaxed* edge-matching criterion. Note that the error bars depict the 95% credible interval if the successful transfer was modeled as a Bernoulli distribution.



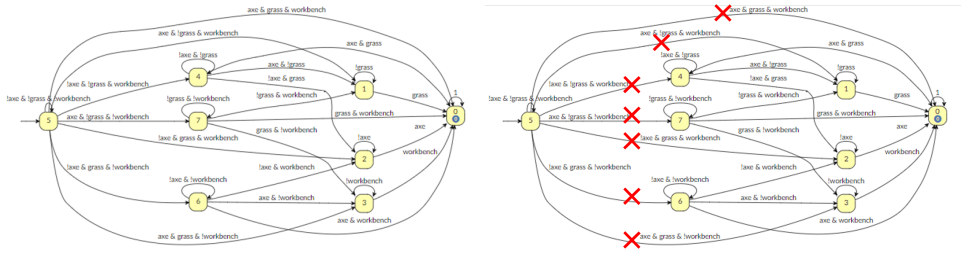
(a) Constrained (b) Relaxed

Figure 3: Figure 3a depicts the success rate of the agent trained on *Soft* training sets of various sizes using LTL-Transfer with the *constrained* edge-matching criterion when transferring to test sets of various specifications types. Figure 3b depicts the success rates with the *relaxed* edge-matching criterion. Note that the error bars depict the 95% credible interval if the successful transfer was modeled as a Bernoulli distribution.



(a) Constrained (b) Relaxed

Figure 4: Figure 4a depicts the success rate of the agent trained on *Strictly Soft* training sets of various sizes using LTL-Transfer with the *constrained* edge-matching criterion when transferring to test sets of various specifications types. Figure 4b depicts the success rates with the *relaxed* edge-matching criterion. Note that the error bars depict the 95% credible interval if the successful transfer was modeled as a Bernoulli distribution.



(a) (b)

Figure 5: Figure 5a depicts the reward machine for the the specification $\varphi_{test} = \mathbf{F}workbench \wedge \mathbf{F}grass \wedge \mathbf{F}axe$, as well as all feasible edges matched by the *Relaxed* criterion. Note that all the edges have at least one matched transition-centric option for the *Relaxed* criterion. Figure 5b depicts the edges that do not have a compatible transition-centric option for the *Constrained* edge matching criterion.

1.4 Example specifications

Here we provide the specifications and the interpretations of three formulas each from the *Hard*, *Soft*, *Strictly Soft*, *No Orders*, and *Mixed* formula types. Note that the training set contained 50 formulas each (new formulas were added incrementally when varying the training set size), and the test set contained 100 formulas each.

Hard: Example formulas belonging to the *Hard* dataset are as follows:

1. $\mathbf{F}wood \wedge \mathbf{F}axe \wedge \neg wood\ \mathbf{U}\ grass \wedge \neg grass\ \mathbf{U}\ workbench \wedge \neg workbench\ \mathbf{U}\ bridge$: Visit $bridge$, $workbench$, $grass$, $wood$, and axe. Ensure that $bridge$, $workbench$, $grass$, $wood$ in that particular order. Objects later in the sequence cannot be visited before the prior objects.

2. $\mathbf{F}workbench \wedge \mathbf{F}factory \wedge \mathbf{F}iron \wedge \mathbf{F}shelter \wedge \neg factory\ \mathbf{U}\ axe$: Visit $workbench$, $factory$, $iron$, $shelter$, and axe. Ensure that $factory$ is not visited before axe.

3. $\mathbf{F}toolshed \wedge \mathbf{F}bridge \wedge \mathbf{F}factory \wedge \mathbf{F}axe \wedge \neg bridge\ \mathbf{U}\ wood$: Visit $toolshed$, $bridge$, $factory$, axe, and $wood$. Ensure that $bridge$ is not visited before $wood$.

Soft: Examples belonging to the *Soft* dataset are as follows:

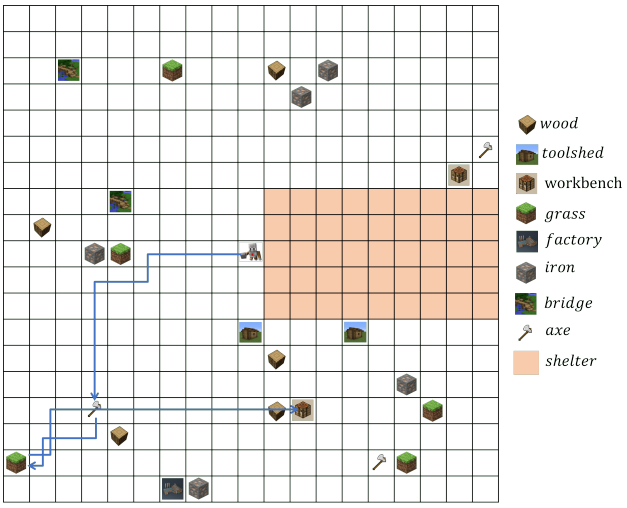


Figure 6: Trajectory executed by the agent using LTL-Transfer on the specification $\varphi_{test} = \mathbf{F}workbench \wedge \mathbf{F}grass \wedge \mathbf{F}axe$.

1. $\mathbf{F}(bridge \wedge \mathbf{F}(factory \wedge \mathbf{F}(iron \wedge \mathbf{F}shelter)))$: Visit $bridge$, $factory$, $iron$, and $shelter$ in that sequence. The objects later in the sequence may be visited before the prior objects, provided that they are visited at least once after the prior object has been visited.
2. $\mathbf{F}workbench \wedge \mathbf{F}(factory \wedge \mathbf{F}grass)$: Visit the $workbench$, $factory$, and $grass$: Visit $grass$ at least once after visiting the $factory$.
3. $\mathbf{F}(axe \wedge \mathbf{F}factory) \wedge \mathbf{F}workbench$: Visit axe, $factory$, and $workbench$. Ensure that $factory$ is visited at least once after axe is.

Strictly Soft: Examples belonging to the *Strictly Soft* dataset are identical to the *Soft* specifications, except they do not allow for simultaneous satisfaction of multiple sub-tasks. The subtasks in sequence must occur strictly temporally after the prior subtask. This is enforced using $\mathbf{XF}a$ instead of $\mathbf{F}a$.

No Orders: These specifications only contain a list of subtasks to be completed. No temporal orders are enforced between the various subtasks.

Mixed: Examples belonging to the *Mixed* dataset are as follows:

1. $\mathbf{F}toolshed \wedge \mathbf{F}factory \wedge \neg toolshed\ \mathbf{U}\ shelter \wedge \mathbf{F}(grass \wedge \mathbf{F}bridge)$: Visit the $toolshed$, $factory$, $shelter$, $grass$, and $bridge$. Ensure that $toolshed$ is not visited before the shelter, and $bridge$ is visited at least once after $grass$.
2. $\mathbf{F}grass \wedge \neg grass\ \mathbf{U}\ toolshed \wedge \mathbf{F}(factory \wedge \mathbf{XF}workbench)$: Visit $grass$, $toolshed$, $factory$, and $workbench$. Ensure that $grass$ is not visited before $toolshed$, and $workbench$ is at least visited once strictly after $factory$.
3. $\mathbf{F}iron \wedge \neg iron\ \mathbf{U}\ toolshed \wedge \mathbf{F}(shelter \wedge \mathbf{XF}wood)$: Visit $iron$, $toolshed$, $shelter$, and $wood$. Ensure that $iron$ is not visited before $toolshed$, and $wood$ is at least visited once strictly after $shelter$.

1.5 Robot demonstrations

We demonstrated LTL-Transfer on Spot [1], a quadruped mobile manipulator, in a household environment, as shown in Figure 7, where the robot can fetch and deliver objects while navigating through the space.

LTL-Transfer first trained policies to solve 2 training tasks $\Phi_{train} = \{\neg desk_a\ \mathbf{U}\ book \wedge \mathbf{F}desk_a,\ \neg desk_b\ \mathbf{U}\ juice \wedge \mathbf{F}desk_b\}$ in simulation. Then we demonstrated the zero-shot transfer capability of LTL-Transfer on a set of test tasks, as shown in Table 1. The robot can complete test tasks that it is expected to succeed, as shown in an example video [2]. For the test tasks that it is

[2] Video: `https://youtu.be/FrY7CWgNMBk`

expected to fail, the robot as expected does not start execution because LTL-Transfer does not produce a feasible path through the reward machine graph given the transition-centric options learned from training tasks Φ_{train}.

The state space of the household environment includes locations of the robot and 4 objects, i.e., 2 desks, a book on a bookshelf and a juice bottle on a kitchen counter. The robot can move in 4 cardinal directions deterministically, and an invalid movement does not change the robot's position. The robot performs the pick action after it moves to the grid cell representing $book$ or $juice$. We finetuned an off-the-shelf object detection model [3] to determine the grasp point from an RGB image by selecting the center point of the most confident bounding box over the target object. The robot performs the place action after it moves to the grid cell representing $desk_a$ or $desk_b$ at the end of a trajectory by an option policy while carrying an object, i.e., a book or a juice bottle.

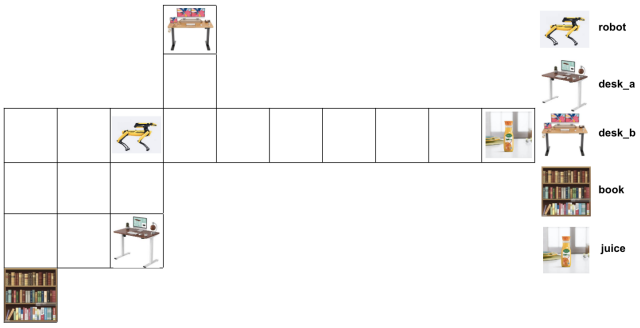


Figure 7: The map of the household environment and the corresponding propositions used for the robot demonstrations.

Table 1: 10 Test LTL Tasks for Robot Demonstrations.

Test LTL Task	**Expected to**
$\mathbf{F}(book \wedge \mathbf{F}(desk_a \wedge \mathbf{F}(juice \wedge \mathbf{F}desk_a)))$	succeed
$\mathbf{F}(book \wedge \mathbf{F}(desk_a \wedge \mathbf{F}(juice \wedge \mathbf{F}desk_b)))$	succeed
$\mathbf{F}(book \wedge \mathbf{F}(desk_b \wedge \mathbf{F}(juice \wedge \mathbf{F}desk_a)))$	succeed
$\mathbf{F}(book \wedge \mathbf{F}(desk_b \wedge \mathbf{F}(juice \wedge \mathbf{F}desk_b)))$	succeed
$\mathbf{F}(juice \wedge \mathbf{F}(desk_a \wedge \mathbf{F}(book \wedge \mathbf{F}desk_a)))$	succeed
$\mathbf{F}(juice \wedge \mathbf{F}(desk_a \wedge \mathbf{F}(book \wedge \mathbf{F}desk_b)))$	succeed
$\mathbf{F}(juice \wedge \mathbf{F}(desk_b \wedge \mathbf{F}(book \wedge \mathbf{F}desk_a)))$	succeed
$\mathbf{F}(juice \wedge \mathbf{F}(desk_b \wedge \mathbf{F}(book \wedge \mathbf{F}desk_b)))$	succeed
$\neg desk_a \mathbf{U}\ book \wedge \neg juice \mathbf{U}\ desk_a \wedge \neg desk_b \mathbf{U}\ juice \wedge \mathbf{F}desk_b$	fail
$\neg desk_b \mathbf{U}\ juice \wedge \neg book \mathbf{U}\ desk_b \wedge \neg desk_a \mathbf{U}\ book \wedge \mathbf{F}desk_a$	fail

References

[1] Boston Dynamics. Spot® - the agile mobile robot. `https://www.bostondynamics.com/products/spot`.

[2] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL `https://doi.org/10.7717/peerj-cs.103`.

[3] Hongkun Yu, Chen Chen, Xianzhi Du, Yeqing Li, Abdullah Rashwan, Le Hou, Pengchong Jin, Fan Yang, Frederick Liu, Jaeyoun Kim, and Jing Li. TensorFlow Model Garden. `https://github.com/tensorflow/models`, 2020.