ANYTOD: A Programmable Task-Oriented Dialog System

Anonymous ACL submission

Abstract

We propose ANYTOD, an end-to-end, zero-shot task-oriented dialog (TOD) system capable of handling unseen tasks without task-specific training. We view TOD as a program executed by a language model (LM), where program logic and ontology is provided by a designer as a schema. To enable generalization to unseen schemas and programs without prior training, ANYTOD adopts a neuro-symbolic approach. A neural LM keeps track of events occurring during a conversation and a symbolic program implementing the dialog policy is executed to recommend next actions ANYTOD should take. This approach drastically reduces data annotation and model training requirements, addressing the enduring challenge of rapidly adapting a TOD system to unseen tasks and domains. We demonstrate state-of-the-art results on STAR (Mehri and Eskenazi, 2021), ABCD (Chen et al., 2021) and SGD (Rastogi et al., 2020) benchmarks. We also demonstrate strong zero-shot transfer ability in low-resource settings, such as zero-shot on MultiWOZ (Budzianowski et al., 2018a). In addition, we release STARv2, an updated version of the STAR dataset with richer annotations, for benchmarking zero-shot end-to-end TOD models.[1]

1 Introduction

An enduring challenge in building and maintaining task-oriented dialog (TOD) systems is efficiently adapting to a new task or domain. For instance, if we were to add the ability to book flight tickets to an existing system that can only handle booking train tickets, this requires new conversations about flight booking to be manually collected and labelled, as well as retraining of natural language understanding (NLU) and policy models. These data efficiency and scaling problems compound for multi-task TOD systems, as each task may have its own bespoke ontology and policy.

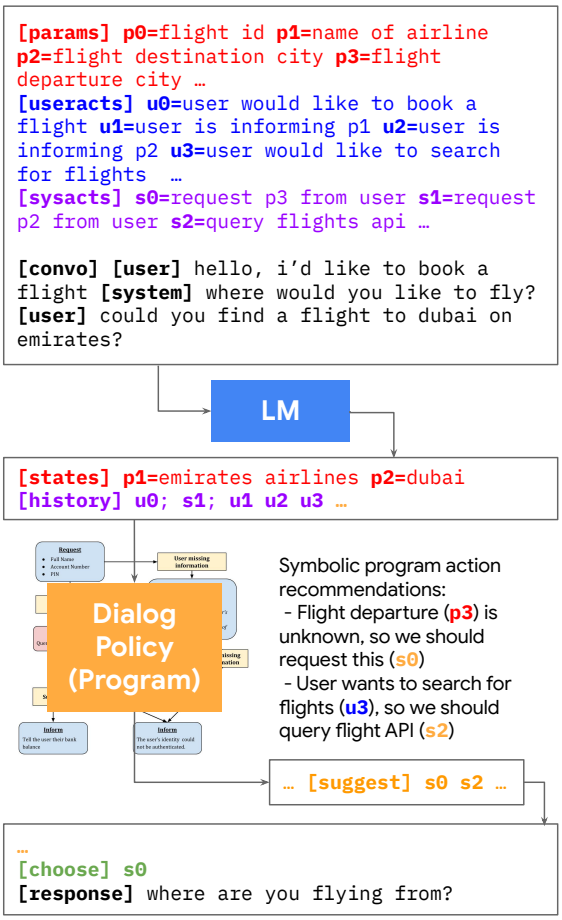

Figure 1: An overview of the ANYTOD system. A LM conducts zero-shot state and action tracking with respect to a provided schema, abstracting it into a sequence of symbols. A program that executes the dialog policy then recommends which actions to take based on the states sequence, the LM then chooses a single final action and generating a response.

To tackle this problem, we propose ANYTOD, an end-to-end TOD system that can be *programmed* to support unseen tasks or domains without prior training, significantly speeding up the TOD design process by easing data collection and training requirements. To the best of our knowledge, ANY-

[1] The STARv2 dataset will be released soon.

TOD is the first end-to-end TOD system capable of zero-shot transfer. To this end, we view TOD as a *program* that a language model (LM) must execute throughout a conversation, and can rely on to provide guidance. Any predefined task policy, implemented as a program, can be used to control ANYTOD, allowing arbitrary business logic to be executed for a specific task. To demonstrate the efficacy of this paradigm, we experiment with the STAR (Mehri and Eskenazi, 2021), ABCD (Chen et al., 2021), SGD (Rastogi et al., 2020) and MultiWoZ (Eric et al., 2019) benchmarks. Not only does ANYTOD achieve state-of-the-art results in full-shot settings, it also achieves high accuracy in zero-shot setups.

Overview of ANYTOD To adhere to a given program, ANYTOD adopts a neuro-symbolic approach (Figure 1). A neural LM is trained for zero-shot dialog state tracking (DST) and action state tracking (AST), abstracting both states and actions into a sequence of symbols. To support zero-shot, we follow the schema-guided paradigm advocated by Rastogi et al. (2020), and provide a *schema* to the LM as contextual information, describing all parameters and actions that should be tracked in natural language. By training on a large corpus of diverse schemas, the LM generalizes to arbitrary and unseen schemas (Lee et al., 2021; Zhao et al., 2022). A schema should also provide a symbolic program that declares the task logic, which is executed to *recommend* possible next actions the agent can take, conditioned on the current dialog states. These recommendations are then reincorporated into the LM, which selects a single next action prediction (NAP), and generates a response. Note that the symbolic program forces ANYTOD to consider a dialog policy explicitly, driving zero-shot transfer onto unseen policies and allowing arbitrarily complex business logic to be employed. However, the program's recommendations are only considered as guidelines, and it is up to the LM to make a final decision on the NAP.

STARv2 We also introduce STARv2, an improved version of the STAR dataset (Mosig et al., 2020). The original STAR dataset is very valuable for benchmarking zero-shot dialog policy and NAP across a diverse set of tasks or domains, through following a provided *policy graph* that outlines the intended flow of a conversation. However, the original dataset made following these policy graphs difficult, due to its lack of training data for DST and AST. Moreover, we found that the schema entity descriptions provided by the original dataset were not intuitive enough to truly support zero-shot DST and AST. To resolve these limitations, the STARv2 dataset provides new belief state and action state annotations to the STAR dataset, as well as more intuitive natural language descriptions for many schema elements. In Section 4.2, we show that these changes facilitate stronger zero-shot DST and AST. However, the ground truth NAP on each system turn is left untouched, allowing direct comparison to results trained on the original STAR dataset. We hope that STARv2 can serve as a new benchmark for TOD systems and drive further research for zero-shot TOD.

2 Related Work

Zero-shot Task-oriented Dialog Fueled by the difficulty of adapting existing TOD systems to new tasks/domains, zero-shot TOD systems have recently seen increasing interest. Much of this work has been on zero-shot DST, with the primary approach being characterizing parameters through names (Wu et al., 2019) or descriptions (Lin et al., 2021; Lee et al., 2021; Zhao et al., 2022). Another approach has been through in-context finetuning (Shah et al., 2019; Gupta et al., 2022), in which a labeled exemplar conversation is given as a prompt to a LM. Mi et al. (2021) demonstrated a more comprehensive approach, including task instructions, constraints, and prompts. In general, these results follow the schema-guided paradigm advocated by Rastogi et al. (2020); Mosig et al. (2020).

By contrast, there are fewer results on zero-shot dialog policy (AST and NAP). To the best of our knowledge, the only result is SAM (Mehri and Eskenazi, 2021), which aligns an LM for an unseen dialog policy by following an explicit policy graph. While similar to the policy graph execution we demonstrate in ANYTOD, there are two differences. First, SAM lacks supervised training on DST and AST, and relies on ground truth NAP only, forcing user state and action tracking to be inextricably linked with the final system action prediction, hurting its ability to generalize to arbitrary policy graphs. Second, SAM is a classification model limited to NAP, and unlike ANYTOD, cannot support DST or natural language generation (NLG). Indeed, we show that ANYTOD is empirically more powerful than SAM in Section 4.2.

To the best of our knowledge, no method has yet to combine zero-shot DST, AST, and NAP into an end-to-end TOD system. All existing end-to-end TOD systems (Hosseini-Asl et al., 2020; He et al., 2021; Yang et al., 2020; Peng et al., 2020) are trained and evaluated on the popular MultiWOZ dataset (Eric et al., 2019). As a result, these systems are only aware of the policy for MultiWOZ, and are not robust to arbitrary/unseen policies. In contrast, AnyTOD can generalize to arbitrary policies, and we demonstrate strong performance on MultiWOZ without prior training (Section 4.4).

TOD as Programming Historically, most TOD approaches use an explicit plan-based dialog policy module (Rich and Sidner, 1998; Ferguson and Allen, 1998; Bohus and Rudnicky, 2009). However, the NLU models powering these TOD systems are tightly coupled to a specific plan, and must be retrained for even slight changes to the plan. In contrast, ANYTOD enables zero-shot dialog policy by training NLU models to be robust to arbitrary programs as policies. Further, ANYTOD uses the program as contextual information to NLU, and refines its NAP with respect to the conversation, belief state, and action history instead of simply accepting the plan's dictated next action(s).

Recent work has also focused on discovering structure within conversations i.e. a latent schema, policy graph, or program (Shi et al., 2019; Yu et al., 2022; Xu et al., 2020). Notably, SMCalFlow (Machines et al., 2020) constructs "dataflow graphs" from a conversation, parsing semantic intents into executable programs. Cheng et al. (2020); Shin et al. (2021) further explore this setup. However, these aim to manipulate an external API/database instead of controlling the agent's behavior.

Beyond the scope of TOD, there has been some work in general neuro-symbolic programming with LMs, in which an LM is influenced by the results of a symbolic system. Nye et al. (2021) demonstrated a symbolic reasnoning module that accepts or rejects the logical consistency of generations from a neural LM. Lu et al. (2020) explored using predicated logic constraints to control lexical aspects from the generation of an LM. However, ANYTOD is the first application of such an approach to a practical TOD setting.

3 Methodology

3.1 The ANYTOD System

An overview of the ANYTOD system is presented in Fig. 1. We decompose ANYTOD into three steps, and describe each step in detail below:

1. Schema and program construction: A designer constructs a schema for ANYTOD to characterize the ontology of a specific task, as well as a policy graph that declares the task logic.

2. Zero-shot DST and AST: A LM performs zero-shot DST and AST with reference to the schema, without task-specific training.

3. Program execution and NAP: The predicted states and action history are passed to the schema program, which upon execution recommends preferred system actions to the agent. These actions are resent to the LM, which predicts the final system action(s) conditioned on these recommendations, conversation history, and belief states.

Schema Construction The designer is required to construct a schema defining a task's ontology, and provide a program describing business logic. This is the *only* thing ANYTOD requires from the designer. For example suppose the designer is creating a flight booking chatbot, they must define the parameters to be tracked (e.g. "flight id", "name of the airline"), and enumerate possible actions the user and agent can take ("user saying they would like to search for flights", "agent should query flight booking api"). Following the schema-guided paradigm advocated in Rastogi et al. (2020), each element in this schema is characterized by a short natural language description, allowing the LM to understand its meaning and facilitate zero-shot transfer. The schema program can be considered as a function that takes in predicted belief states and actions, and dictate possible NAPs following explicit symbolic rules. Examples can be seen in Section A.1. In general, this program should infer agent actions in response to user behavior (e.g. "if user wants to search for flights, query the flight search api").

Zero-shot DST and AST Adaptation to novel tasks without training data critically hinges on an LM performing zero-shot DST and AST. For this purpose, we adopt and extend the D3ST approach (Zhao et al., 2022) due to its flexibility in zero-

shot state and action tracking. Specifically, D3ST conducts zero-shot DST in the following way. Let $p_0, ...p_n$ be the parameters defined in the schema, and let $\text{desc}(p_i)$ denote a parameter's natural language description. Then, construct a parameter context string

$$\texttt{[params]}\ \texttt{p}0\texttt{=}\text{desc}(p_0)\ \ \texttt{...}\ \ \texttt{p}n\texttt{=}\text{desc}(p_n)$$

Note that the strings $\texttt{p}0$, ..., $\texttt{p}n$ are used as indices. Similar context strings are generated for actions for AST. These context strings are concatenated with the entire conversation history, forming the input to the LM. This input is contextualized by the schema information, allowing the LM to refer to the schema, and enabling zero-shot transfer. The target string contains the conversation belief state and history of actions at each turn of the conversation, both in a parseable format. Let $p_{i_0}, \ldots, p_{i_m}$ be the active parameters in the conversation, with corresponding values $v_{i_0}, \ldots, v_{i_m}$. The belief state is represented as

$$\texttt{[state]}\ \texttt{p}i_0\texttt{=}v_{i_0}\texttt{;}...\texttt{;}\ \texttt{p}i_m\texttt{=}v_{i_m}$$

Note that inactive slots do not appear in the belief state string. In ANYTOD D3ST is naturally extended to perform zero-shot AST. Note that D3ST's original formulation in Zhao et al. (2022) was limited to DST, but, in principle, D3ST supports tracking arbitrary events that occur during a conversation, as long as their descriptions are provided. For AST, we build an target string consisting of a history of actions that were active at each turn of the conversation. Let $\texttt{u}j$ and $\texttt{s}k$ be the format of D3ST indices for user and system actions. Then, an action history string may look like

[history] u0 u9; s2; u1; s3; ...

This denotes that, on the first turn, the user was performing user actions **u0** and **u9**. On the second turn, the system was performing system action **s2**, and so on. Note that the active actions for each turn are separated by a **;** character.

Program Execution The LM's predicted belief states and action history are then parsed and passed to the schema program. This program should execute the dialog policy and control ANYTOD, by recommending possible NAPs. Section A.1 showcases some example programs for STARV2 tasks. In the example shown in Figure 1, the current conversation state ("user would like to search for flights to Dubai with Emirates") satisfies multiple dependency rules ("since the user would like to search for flights, query the flight search api" and "since the user has not provided their flight departure location, ask the user for it"). These system actions are then passed back to the LM as a string of system action indices.

[recommend] s0 s2

Finally, given the policy graph's recommended actions as extra conditional information, the LM makes predictions about NAP with respect to the conversation, previously predicted belief states and action history. A response is also generated following the action prediction.

[selected] s2 [response] hello!

Note that the selected action need not be one of the actions recommended from the policy graph output, because actual conversations may not rigorously follow the predefined business logic, and violations are common. This step allows ANYTOD to "softly" execute the policy graph, balancing between the model's belief before and after receiving recommendations.

Zero-shot transfer ANYTOD's zero-shot transfer ability is enabled by a combination of two design considerations. The first is the LM's description-driven state and event tracking. Since this schema information is provided as context, if this LM is trained on a corpus of diverse schemas, it learns to make predictions by "reading" and understanding the schema descriptions. This leads to robustness on ANYTOD's state and event tracking for unseen schemas, as shown in Zhao et al. (2022). Moreover, ANYTOD facilitates zero-shot policy transfer by executing the provided policy graphs as explicit rules, and by similarly training the LM with a large number of policy graphs when selecting a recommended system action.

3.2 The STARV2 Dataset

To train ANYTOD, we construct STARV2, an updated version of STAR with new ground truth belief state and action annotations, supporting supervised training on DST and AST. These annotations were generated from few-shot training with D3ST (Zhao et al., 2022). We first train D3ST on the SGD dataset, then continue finetuning on a few hand-labeled conversations from STAR.[2] While not the

[2] 4 conversations were labeled from each task in STAR.

focus of this paper, the labeling of STARv2 demonstrates the use of few-shot D3ST in labeling unlabeled conversations on new tasks/domains.

Further, STARv2 adds new natural language descriptions for actions in STAR schemas. Prior work on STAR (Mosig et al., 2020; Mehri and Eskenazi, 2021) leverages template utterances as schema descriptions, which we qualitatively found to not fully outline the complexity of actions e.g., the action `user_weather_inform_city` has a template utterance of just `[CITY]`. STARv2 provides `user is informing city` as a more natural action description. We show in Section 4.2 that these actions improve zero-shot AST.

4 Experiments

4.1 Setup

Datasets We demonstrate ANYTOD's power in zero-shot settings on the following datasets:

STAR and STARv2: As described in Section 3.2, we upgrade the original STAR (Mehri and Eskenazi, 2021) dataset to STARv2. The dataset has 24 tasks across 13 domains, many tasks requiring the model to adhere to a novel policy, providing an important zero-shot AST and NAP benchmark.

ABCD (Chen et al., 2021): The design of the ABCD dataset follows a realistic setup, in which an agent's actions must be balanced between the customer's expressed desires and the constraints set by task policies. It is thus a natural fit for the AnyTOD framework for both training and evaluation.

SGD (Rastogi et al., 2020): SGD is another schema-guided dataset in which schema elements are provided with natural language descriptions to facilitate task transfer. It contains 45 domains and was generated via simulation. Thus, the agent actions and responses follow pre-defined task logic.

MultiWOZ (Budzianowski et al., 2018b): MultiWOZ is the standard dataset for benchmarking TOD models. It contains 7 domains and was generated through Wizard-of-Oz (Kelley, 1984) data collection, leading to natural conversations.

Training Our implementation is based upon the open-source T5X codebase (Roberts et al., 2022) initialized with the public T5 1.1 checkpoints[3] as the LM backend. We update the LM code to execute a schema program and reincorporate the results before making the final NAP, as described in Section 3.1. We experimented on two T5 sizes: base (250M parameters, trained on 16 TPUv3 chips (Jouppi et al., 2017)) and XXL (11B parameters, trained on 64 TPUv3 chips). We otherwise adopt the default T5X finetuning hyper-parameter settings throughout our experiments.

4.2 Results on STAR

Table 1 shows ANYTOD results on the STARv2 dataset on the full-shot and zero-shot domain transfer settings, with both "happy" and "unhappy" conversations. In full-shot, models train on 80% of conversations across all tasks, and evaluate on the remaining 20%. The zero-shot domain setting is a leave-one-out cross-validation across the STARv2 dataset's 13 domains, evaluating quality on an unseen schema in a completely novel domain. The following metrics are used in our report: joint goal accuracy (JGA) to measure DST, user action F1 (UaF1) to measure AST, system action F1 (SaF1) to measure NAP, and response BLEU.[4]

Each STAR task schema defines the intended dialog policy by providing a *policy graph*, where nodes describe conversation actions, and edges connect subsequent actions. An ANYTOD program (Figure A.2) is implemented to recommend next actions with respect to this policy graph.

Two baselines are used for comparison: BERT+S (Mosig et al., 2020) and SAM (Mehri and Eskenazi, 2021), both of which add a policy graph following module for zero-shot transfer to unseen schema. Note that, though these models were trained on the original STAR data, their SaF1 results are directly comparable to ANYTOD trained on STARv2 on NAP (SaF1), as these ground truth labels were left untouched. However, ANYTOD has additional training supervision on AST and DST due to STARv2's new annotations. For a fair comparison with SAM, we also report results on SAM-User, a modified version of SAM trained on STARv2 that also includes supervised training on user annotations.[5] Note that both BERT+S and SAM are based on BERT-base (110M parameters), comparable to T5 base (220M parameters).

Main Result The primary results for ANYTOD

[3] https://github.com/google-research/text-to-text-transfer-transformer

[4] See Section A.4 for details on calculating these metrics.

[5] See Section A.5 for implementation details.

BASE/XXL are given in Table 1. For conciseness, we shorten ANYTOD to AT. As an ablation, we also report results with AT-NOREC, which removes the policy graph guidance from the ANYTOD method by recommending no system actions. In the full-shot setting, both ANYTOD and -NOREC, along with reported baselines, achieve very high SaF1. This is due to direct supervised training on NAP removing the need for program guidance. However, we see a huge gap between ANYTOD and -NOREC in the zero-shot setting; the guidance from the program becomes necessary — we see 60.6 vs. 55.8 SaF1 at BASE, and 68.0 vs. 62.3 SaF1 at XXL. Moreover, ANYTOD XXL has zero-shot performance comparable to that of full-shot, with 75.4 SaF1 at XXL.

Effect of Natural Language Descriptions As mentioned in Section 3.2, STARV2 provides new natural language descriptions that better characterize the actions within STAR. Our main result AT BASE/XXL takes advantage of these new descriptions, but to see the impact of these descriptions, we train AT-TMPL on the original template utterances. On BASE we see little difference between descriptions and templates, but a sizeable improvement in using descriptions appears on XXL, with a larger LM that is better at NLU. This evidences that more intuitive natural language descriptions help ANYTOD understand task semantics better and perform zero-shot transfer.

ANYTOD vs. baselines To compare against available results on STARV2, we compare AT-TMPL BASE against SAM-User. Both results use template responses provided by STAR, and additionally trained with the new DST and AST annotations in STARV2. However, we see far stronger performance with ANYTOD than with SAM or SAM-User, due to the flexibility provided by the program execution ability demonstrated by ANYTOD, and enabled by supervised training on DST and AST. SAM is not suited to use these contextual signals, likely due to no attention between schema elements and conversation and a rigid classification architecture unsuitable for multiple losses.

Multitask Training with SGD To demonstrate further robustness for ANYTOD, we also report ANYTOD-SGD, which jointly trains with SGD as a multitask training dataset. SGD includes a large number of tasks, each defined by a schema with highly diverse parameters and actions. The -SGD results in Table 1 show that at BASE, SGD multitask training improves both DST (61.9 → 66.1 JGA), AST (72.1 → 74.3 UaF1), and by extension NAP (60.6 → 61.3 SaF1). A similar but smaller improvement is seen on XXL, suggesting that the larger LM may not need more diverse training owing to its better language understanding.

Model	JGA	UaF1	SaF1	BLEU
BERT+S	-	-	74.9	-
SAM	-	-	71.5	-
SAM-User	-	-	71.7	-
AT-NOREC BASE	81.5	83.8	73.3	72.8
AT-TMPL BASE	82.9	84.6	70.6	72.7
AT BASE	82.4	84.1	70.7	72
AT-NOREC XXL	85.6	86.4	75.4	76.4
AT-TMPL XXL	85.1	82.5	71.3	75.8
AT XXL	85.7	84.7	73.3	73.5

(a) Full-shot results on STARV2.

Model	JGA	UaF1	SaF1	BLEU
BERT+S	-	-	32.3	-
SAM[6]	-	-	51.2	-
SAM-User	-	-	44.4	-
AT-NOREC BASE	57.8	71	55.8	32.4
AT-TMPL BASE	62.2	74	61.9	56
AT BASE	61.9	72.1	**60.6**	34.3
AT-SGD BASE	66.1	74.3	61.3	34.4
AT-PROG BASE	61.9	72.1	61.0	34.4
AT-PROG+SGD BASE	66.1	74.3	61.9	34.6
AT-NOREC XXL	72.7	80	62.3	41.8
AT-TMPL XXL	66.8	72.9	60.8	52.9
AT XXL	74.8	79.2	**68.0**	44.3
AT-SGD XXL	75.8	80.9	68.5	43.9
AT-PROG XXL	74.4	79.3	68.4	44.9
AT-PROG+SGD XXL	75.7	81.4	70.7	44.2

(b) Zero-shot domain results on STARV2.

Model	Bank	Trip	Trivia
AT XXL	54.3	52.4	73.8
AT-SGD XXL	53.1	51.5	81.1
AT-PROG XXL	61	60.8	73.7
AT-PROG+SGD XXL	65	62.9	86.3

(c) SaF1 on STARV2 programming tasks..

Table 1: Results on STARV2. For compactness we show just UaF1 and SaF1 here — see Section A.2 for a complete table. For clarity, we bold SaF1 results for ANYTOD BASE/XXL, our key result.

Complex Program Logic STARV2 is also a good testbed for complex zero-shot task adaptation, as it includes some tasks which are more complex than simple policy-graph following, specifically the `bank`, `trivia`, and `trip` domains. For instance, the `trivia` task requires the agent to ask the user a trivia question and extract their answer. Different

[6]Note that this SAM zero-shot domain SaF1 differs from the original 55.7 from Mehri and Eskenazi (2021). See Section A.3 for more details.

Model	JGA seen	JGA unseen	SaF1 seen	SaF1 unseen
AT-NOREC BASE	89.0	58.5	89.8	83.4
AT BASE	89.9	62.4	89.8	86.1
AT-NOREC XXL	94.8	80.2	92.1	87.2
AT XXL	94.8	82.2	91.3	88.9

Table 2: ANYTOD JGA, SaF1 on SGD test set.

system actions must be taken by the agent depending on whether or not the user's answer is correct. This logic is not captured by the provided policy graph alone, requiring more complex logic. ANYTOD is suitable for this problem, as we need only to construct a program implementing this logic. These programs are shown in Section A.1.

We report results with these programs in Table 1 under the -PROG name. There is a clear win on zero-shot domain SaF1 when averaged over all domains, with a very high 70.7 SaF1 on -PROG+SGD XXL, narrowing the gap with the full-shot 75.4 SaF1. When examining the complex tasks tasks individually (Table 1c), the win on NAP is even more apparent. The only exception is AT XXL on `trivia`, which has little difference with or without the program. In general however, the guidance provided by this specialized program is necessary for higher-level logic in the dialog policy, since the policy graph does not specify enough information to approach the task in zero-shot.

4.3 Results on ABCD and SGD

We conduct similar experiments on Action State Tracking (AST) (metric: joint action accuracy or JAA) on ABCD (Chen et al., 2021) and DST and NAP (metrics: JGA and SaF1 respectively) on SGD (Rastogi et al., 2020) datasets.

ABCD contains 10 flows, each describing the business logic for handling a customer request, which are relatively similar to each other. We report full-shot results by training and evaluating on all flows, and zero-shot results where the model is trained on one randomly sampled flow and evaluated on all other nine flows. The SGD test set consists of 21 services, 15 of these not seen during training. The dataset is generated via simulation with a generalized policy graph (shared across all services) encoding dialog act transitions. The per-service policy graphs are then constructed by inserting intents and slots and, as a result, end up similar.

Tables 2 and 3 and show ANYTOD results on SGD and ABCD respectively. For both datasets on both full-shot and zero-shot setups we generally see an improvement on action prediction using policy guidance, achieving state-of-the-art results for ABCD. However, the gain is not as large as STARV2, as the task policies are not as diverse. Even without explicit policy guidance, features from different tasks in ABCD/SGD can transfer to each other. Notably, policy guidance helps more on the one-flow setup for ABCD and unseen services for SGD, further establishing the efficacy of policy guidance on unseen setups, even if related.

4.4 Zero-shot Results on MultiWOZ

To demonstrate the generalizability of the ANYTOD system, we demonstrate zero-shot transfer results on the end-to-end MultiWOZ 2.2 (Zang et al., 2020) benchmark, a popular dataset for TOD research. In this case, ANYTOD-XXL is trained on the SGD dataset, and then evaluated on MultiWOZ in zero-shot with a small policy program (Section A.6). Responses from ANYTOD were constructed using the template utterance approach from Kale and Rastogi (2020). We compare against SOLOIST (Peng et al., 2020) and Mars (Sun et al., 2022), two end-to-end TOD models directly trained on MultiWOZ with supervision. Results are shown in Table 5, with metrics reported by the MultiWOZ eval script (Nekvinda and Dusek, 2021). Although no training examples from MultiWOZ was used at all, ANYTOD demonstrates strong JGA, Inform, and Success comparable to results that do train on MultiWOZ. Note that since we applied templates for response generation, we do not consider BLEU to be important, as the responses are very different from ground truth labels.

5 Analysis

5.1 Impact of Policy Guidance

To see how impactful the recommendations provided by the policy graph are, we reevaluate already finetuned ANYTOD models on the STARV2 zero-shot domain setting, but with changes to the program recommendations during eval. First, to see how dependent ANYTOD is on policy graph guidance, we modify the graph to output no recommendations (denoted as 0REC), forcing the model to do NAP only using the conversation, belief state, and action history. Secondly, we modify the graph to output deliberately bad recommendations (denoted as BADREC), intended to trick the model into choosing an incorrect system action. This was

Model	All Flows	One Flow
RoBERTa	65.8	-
AST-T5-Small	87.9	-
AT-NOREC BASE	90.5	47.4
AT BASE	**90.5**	**48.9**
AT-NOREC XXL	91.6	64.3
AT XXL	**91.9**	**67**

Table 3: JAA on ABCD Action State Tracking (AST) for full-shot (All Flows) and zero-shot transfer (One Flow). The zero-shot JAA is the mean JAA across three experiments.

Model	SaF1
AT BASE	60.6
AT-0REC BASE	31.3
AT-BADREC BASE	25.8
AT XXL	68.0
AT-0REC XXL	39.3
AT-BADREC XXL	35.0

Table 4: STARV2 zero-shot domain SaF1 with BADREC and 0REC.

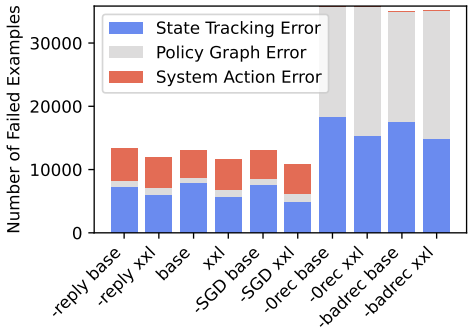

Figure 2: ANYTOD error analysis on STARV2 zero-shot domain.

Model	JGA	Inform	Success	BLEU
SOLOIST	35.9	81.7	67.1	13.6
Mars	35.5	88.9	78.0	19.6
ANYTOD-XXL	30.8	73.9	24.4	3.4

Table 5: Results on MultiWOZ end-to-end benchmark. ANYTOD-XXL is trained on SGD, and evaluated in zero-shot over MultiWOZ. Note we applied templates for response generation, yielding low BLEU in comparison with other models.

Model and Corruption Prob.	All Flows	One Flow
AT BASE, 0	90.5	48.9
AT BASE, 0.4	90.1	48.4
AT BASE, 0.8	89.5	47.4
AT-NOREC BASE, 0	90.5	47.4
AT XXL, 0	91.9	67
AT XXL, 0.4	91.5	66.7
AT XXL, 0.8	91.5	65.9
AT-NOREC XXL, 0	91.6	64.3

Table 6: JAA on ABCD Action State Tracking (AST) with policy corruption. For "one flow", the JAA is averaged across three runs with a randomly selected flow for training.

done by randomly sampling 1-3 system actions other than the ground truth action.

The major drops in SaF1 for both setups shown in Table 4 confirm that the model, while able to predict actions without it, does consider the policy guidance heavily. Notably, 75% and 83% of correct predictions for 0REC and BADREC are actions common to all tasks e.g., `hello` or `query`.

We conduct a similar "policy corruption" experiment on ABCD (Table 6), in which policy graphs for evaluation tasks have a 0%, 40%, and 80% chance of being replaced by graphs from incorrect flows during evaluation. We see a consistent quality drop with increasing probability of corruption for both BASE and XXL.

5.2 Error Analysis

We also analyze ANYTOD errors on STARV2. We classify all incorrect NAPs into three possible error categories: (1) System action error: the program recommends the correct system action, but this was not chosen by the LM, (2) Policy graph error: the predicted belief state and action history are correct, but the program's execution of the policy graph does not recommend the expected system action, and (3) State tracking error: the predicted belief states and action history are incorrect, which leads to incorrect recommendations from the policy graph. Results are shown in Figure 2. In general, we see that the benefit to scaling the LM from BASE to XXL comes from improvements to state and action tracking, which aligns with better DST and AST results on XXL as in Table 1.

6 Conclusion

We proposed ANYTOD, a zero-shot end-to-end TOD system that can be programmed to handle unseen tasks without domain-specific training. ANYTOD adopts a neuro-symbolic approach, in which a LM performs zero-shot DST and AST with respect to a provided schema, and abstracts both into a sequence of symbols. These symbol sequences are then parsed and passed to a program expressing the task policy, which gets executed to make recommendations for the next agent action(s). Agent designers are free to implement arbitrarily complex business logic within ANYTOD to determine its policy on unseen tasks or domains. To demonstrate the value of this approach, we show state-of-the-art results on zero-shot TOD benchmarks, such as STAR, ABCD, SGD and MultiWoZ. For further training and benchmarking zero-shot end-to-end TOD systems, we also release the STARV2 dataset, an improved version of STAR.

Limitations

ANYTOD is a task-oriented dialogue system designed for efficient building of conversational agents with little training data. A large 11B parameter language model (T5) is trained to make generalized structured predictions of dialogue states. A symbolic policy program takes these dialogue states as arguments, and then recommends possible actions ANYTOD should take in response to user behavior. By training on the STARV2 dataset, ANYTOD robustly generalizes to arbitrary and unseen domains for any chatbot policies.

Conversational agents built with ANYTOD are explicitly designed to follow policies predefined by the ANYTOD schema and policy program. As such, ANYTOD is guaranteed to follow predictable and safe behavior when interacting with human users, but is not capable of taking actions outside of the discrete set of actions defined by the schema. As such, we do not intend to use ANYTOD in open-domain, free-form conversation generation scenarios.

While we note that generating free-form natural language responses is possible due to supervised training on ground truth system responses, there is no guarantee that these generated responses are robust on unseen schema. We instead advocate that responses should be built with deterministic templates predefined by agent designers.

Ethics Statement

Models, codebases, and datasets used in this paper follow their respective licenses and terms of use. Moreover, the task-oriented dialogue datasets used in this paper do not contain any personally-identifiable information or offensive content. The code for ANYTOD and the STARV2 dataset will be released upon this paper's publication.

One particular risk with language models is the possible generation of factually incorrect or biased content (Lin et al., 2022; Bender et al., 2021). However, we note that this risk does not apply to ANYTOD, as (1) the language model is trained to make structured predictions that must be parseable by the policy program, and (2) we rely on response templates rather than using free form natural language generation.

References

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big? . In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 610–623, New York, NY, USA. Association for Computing Machinery.

Dan Bohus and Alexander I Rudnicky. 2009. The RavenClaw dialog management framework: Architecture and systems. *Comput. Speech Lang.*, 23(3):332–361.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Ultes Stefan, Ramadan Osman, and Milica Gašić. 2018a. Multiwoz - a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Pawel Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gasic. 2018b. Multiwoz - A large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. *CoRR*, abs/1810.00278.

Derek Chen, Howard Chen, Yi Yang, Alexander Lin, and Zhou Yu. 2021. Action-based conversations dataset: A corpus for building more in-depth task-oriented dialogue systems. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3002–3017, Online. Association for Computational Linguistics.

Jianpeng Cheng, Devang Agrawal, Hector Martinez Alonso, Shruti Bhargava, Joris Driesen, Federico Flego, Shaona Ghosh, Dain Kaplan, Dimitri Kartsaklis, Lin Li, Dhivya Piraviperumal, Jason D Williams, Hong Yu, Diarmuid O Seaghdha, and Anders Johannsen. 2020. Conversational semantic parsing for dialog state tracking.

Mihail Eric, Rahul Goel, Shachi Paul, Adarsh Kumar, Abhishek Sethi, Peter Ku, Anuj Kumar Goyal, Sanchit Agarwal, Shuyang Gao, and Dilek Hakkani-Tur. 2019. MultiWOZ 2.1: A consolidated Multi-Domain dialogue dataset with state corrections and state tracking baselines.

George Ferguson and James F Allen. 1998. TRIPS: An integrated intelligent problem-solving assistant. https://www.aaai.org/Papers/AAAI/1998/AAAI98-080.pdf. Accessed: 2022-12-14.

Raghav Gupta, Harrison Lee, Jeffrey Zhao, Yuan Cao, Abhinav Rastogi, and Yonghui Wu. 2022. Show, don't tell: Demonstrations outperform descriptions for schema-guided task-oriented dialogue. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4541–4549, Seattle, United States. Association for Computational Linguistics.

Wanwei He, Yinpei Dai, Yinhe Zheng, Yuchuan Wu, Zheng Cao, Dermot Liu, Peng Jiang, Min Yang, Fei Huang, Luo Si, Jian Sun, and Yongbin Li. 2021. GALAXY: A generative pre-trained model for Task-Oriented dialog with Semi-Supervised learning and explicit policy injection.

Ehsan Hosseini-Asl, Bryan McCann, Chien-Sheng Wu, Semih Yavuz, and Richard Socher. 2020. A simple language model for task-oriented dialogue.

Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News*, 45(2):1–12.

Mihir Kale and Abhinav Rastogi. 2020. Few-shot natural language generation by rewriting templates. *CoRR*, abs/2004.15006.

J F Kelley. 1984. An iterative design methodology for user-friendly natural language office information applications. *ACM Trans. Inf. Syst. Secur.*, 2(1):26–41.

Chia-Hsuan Lee, Hao Cheng, and Mari Ostendorf. 2021. Dialogue state tracking with a language model using Schema-Driven prompting.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252, Dublin, Ireland. Association for Computational Linguistics.

Zhaojiang Lin, Bing Liu, Seungwhan Moon, Paul Crook, Zhenpeng Zhou, Zhiguang Wang, Zhou Yu, Andrea Madotto, Eunjoon Cho, and Rajen Subba. 2021. Leveraging slot descriptions for Zero-Shot Cross-Domain dialogue StateTracking. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5640–5648, Online. Association for Computational Linguistics.

Ximing Lu, Peter West, Rowan Zellers, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. NeuroLogic decoding: (un)supervised neural text generation with predicate logic constraints.

Semantic Machines, Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, Hao Fang, Alan Guo, David Hall, Kristin Hayes, Kellie Hill, Diana Ho, Wendy Iwaszuk, Smriti Jha, Dan Klein, Jayant Krishnamurthy, Theo Lanman, Percy Liang, Christopher H Lin, Ilya Lintsbakh, Andy McGovern, Aleksandr Nisnevich, Adam Pauls, Dmitrij Petters, Brent Read, Dan Roth, Subhro Roy, Jesse Rusak, Beth Short, Div Slomin, Ben Snyder, Stephon Striplin, Yu Su, Zachary Tellman, Sam Thomson, Andrei Vorobev, Izabela Witoszko, Jason Wolfe, Abby Wray, Yuchen Zhang, and Alexander Zotov. 2020. Task-Oriented dialogue as dataflow synthesis.

Shikib Mehri and Maxine Eskenazi. 2021. Schema-guided paradigm for zero-shot dialog. In *Proceedings of the 22nd Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 499–508, Singapore and Online. Association for Computational Linguistics.

Fei Mi, Yitong Li, Yasheng Wang, Xin Jiang, and Qun Liu. 2021. CINS: Comprehensive instruction for few-shot learning in task-oriented dialog systems.

Johannes E M Mosig, Shikib Mehri, and Thomas Kober. 2020. STAR: A Schema-Guided dialog dataset for transfer learning.

Tomás Nekvinda and Ondrej Dusek. 2021. Shades of bleu, flavours of success: The case of multiwoz. *CoRR*, abs/2106.05555.

Maxwell Nye, Michael Henry Tessler, Joshua B Tenenbaum, and Brenden M Lake. 2021. Improving coherence and consistency in neural sequence models with Dual-System, Neuro-Symbolic reasoning.

Baolin Peng, Chunyuan Li, Jinchao Li, Shahin Shayandeh, Lars Liden, and Jianfeng Gao. 2020. SOLOIST: Building task bots at scale with transfer learning and machine teaching.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. 2020. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8689–8696.

Charles Rich and Candace L Sidner. 1998. COLLAGEN: A collaboration manager for software interface agents. *User Model. User-adapt Interact.*, 8(3):315–350.

Adam Roberts, Hyung Won Chung, Anselm Levskaya, Gaurav Mishra, James Bradbury, Daniel Andor, Sharan Narang, Brian Lester, Colin Gaffney, Afroz Mohiuddin, Curtis Hawthorne, Aitor Lewkowycz, Alex Salcianu, Marc van Zee, Jacob Austin, Sebastian Goodman, Livio Baldini Soares, Haitang Hu, Sasha

Tsvyashchenko, Aakanksha Chowdhery, Jasmijn Bastings, Jannis Bulian, Xavier Garcia, Jianmo Ni, Andrew Chen, Kathleen Kenealy, Jonathan H. Clark, Stephan Lee, Dan Garrette, James Lee-Thorp, Colin Raffel, Noam Shazeer, Marvin Ritter, Maarten Bosma, Alexandre Passos, Jeremy Maitin-Shepard, Noah Fiedel, Mark Omernick, Brennan Saeta, Ryan Sepassi, Alexander Spiridonov, Joshua Newlan, and Andrea Gesmundo. 2022. Scaling up models and data with `t5x` and `seqio`. *arXiv preprint arXiv:2203.17189*.

Darsh J Shah, Raghav Gupta, Amir A Fayazi, and Dilek Hakkani-Tur. 2019. Robust Zero-Shot Cross-Domain slot filling with example values.

Weiyan Shi, Tiancheng Zhao, and Zhou Yu. 2019. Unsupervised dialog structure learning. *CoRR*, abs/1904.03736.

Richard Shin, Christopher Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021. Constrained language models yield few-shot semantic parsers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Stroudsburg, PA, USA. Association for Computational Linguistics.

Haipeng Sun, Junwei Bao, Youzheng Wu, and Xiaodong He. 2022. Mars: Semantic-aware contrastive learning for End-to-End Task-Oriented dialog.

Chien-Sheng Wu, Andrea Madotto, Ehsan Hosseini-Asl, Caiming Xiong, Richard Socher, and Pascale Fung. 2019. Transferable Multi-Domain state generator for Task-Oriented dialogue systems.

Jun Xu, Zeyang Lei, Haifeng Wang, Zheng-Yu Niu, Hua Wu, Wanxiang Che, and Ting Liu. 2020. Discovering dialog structure graph for Open-Domain dialog generation.

Yunyi Yang, Yunhao Li, and Xiaojun Quan. 2020. UBAR: Towards fully End-to-End Task-Oriented dialog systems with GPT-2.

Dian Yu, Mingqiu Wang, Yuan Cao, Izhak Shafran, Laurent El Shafey, and Hagen Soltau. 2022. Unsupervised slot schema induction for task-oriented dialog.

Xiaoxue Zang, Abhinav Rastogi, Srinivas Sunkara, Raghav Gupta, Jianguo Zhang, and Jindong Chen. 2020. MultiWOZ 2.2 : A dialogue dataset with additional annotation corrections and state tracking baselines. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*, pages 109–117, Online. Association for Computational Linguistics.

Jeffrey Zhao, Raghav Gupta, Yuan Cao, Dian Yu, Mingqiu Wang, Harrison Lee, Abhinav Rastogi, Izhak Shafran, and Yonghui Wu. 2022. Description-driven task-oriented dialog modeling.

A Appendix

A.1 ANYTOD Programs

Examples of ANYTOD program implementations for STARv2 can be found in Figures A.2 and A.3.

A.2 Complete Results on STARv2

For compactness, Table 1 showed just UaF1 and SaF1. We also report user action accuracy (UaAcc) and system action accuracy (SaAcc) in Table A.1.

A.3 Corrected SAM Results on Zero-shot Domain

During the development of ANYTOD, we found that the zero-shot domain results reported on SAM in Mehri and Eskenazi (2021) were incorrect. An annotation issue within the STAR dataset set marked some conversations as having an invalid domain; due to how SAM was implemented, these conversations would always be included in the training dataset, even if they were in the evaluation domain. For instance, dialog ID `102` is marked as a `null` domain in the original STAR dataset. Retraining SAM with this issue fixed caused a drop in SaF1, from 55.7 to 51.2. We fix these annotation errors in the STARv2 dataset.

A.4 Calculating STARv2 Metrics

Details in calculating metrics on STARv2 are as follows. For DST, JGA is calculated with an exact match on belief state parameters and values. For AST, we only consider the quality of the most recent turn within the action history prediction. This is always a user turn, which may have multiple user actions active. This may be considered a multilabel classification problem. Then, we calculate UaAcc through exact set match on the predicted user actions at the current turn, as well as weighted multilabel F1 on the predicted user actions. Both SaAcc and SaF1 are calculated as described in Mosig et al. (2020).

A.5 Implementation of Sam-User

To implement supervised training of AST on SAM, we modify the methodology described in Mehri and Eskenazi (2021), which embeds both conversation and schema elements to produce an attention vector p. Here, p_i gives the attention weight between the conversation and the i-th user action of the policy graph. This is then interpreted to be a proxy for

Model	JGA	UaAcc	UaF1	SaAcc	SaF1	BLEU
BERT+S	-	-	-	73.8	74.9	-
SAM	-	-	-	70.4	71.5	-
SAM-User	-	-	-	70.4	71.7	-
AT-NOGUIDE BASE	81.5	74.4	83.8	73.7	73.3	72.8
AT-TMPL BASE	82.9	75.6	84.6	71	70.6	72.7
AT BASE	82.4	75.2	84.1	71.6	70.7	72
AT-NOGUIDE XXL	85.6	78.3	86.4	75.7	75.4	76.4
AT-TMPL XXL	85.1	72.6	82.5	70.7	71.3	75.8
AT XXL	85.7	75.9	84.7	73.8	73.3	73.5

(a) Full-shot results on STARv2.

Model	JGA	UaAcc	UaF1	SaAcc	SaF1	BLEU
BERT+S	-	-	-	29.7	32.3	-
SAM	-	-	-	49.8	51.2	-
SAM-User	-	-	-	53.9	44.4	-
AT-NOGUIDE BASE	57.8	55.4	71	56.1	55.8	32.4
AT-TMPL BASE	62.2	56	74	62.5	61.9	56
AT BASE	61.9	56.6	72.1	61.6	60.6	34.3
AT-SGD BASE	66.1	59.5	74.3	63.5	61.3	34.4
AT-PROG+REPLY BASE	62.7	55.8	73.9	63.1	62.9	56.3
AT-PROG BASE	61.9	56.6	72.1	61.9	61.0	34.4
AT-PROG+SGD BASE	66.1	59.5	74.3	64.2	61.9	34.6
AT-NOGUIDE XXL	72.7	65.9	80	62.3	62.3	41.8
AT-TMPL XXL	66.8	58.9	72.9	60.9	60.8	52.9
AT XXL	74.8	64.6	79.2	68	68.0	44.3
AT-SGD XXL	75.8	67.8	80.9	69.3	68.5	43.9
AT-PROG+REPLY XXL	73.7	61.6	76.6	65.7	66.3	63.6
AT-PROG XXL	74.4	64.7	79.3	68.5	68.4	44.9
AT-PROG+SGD XXL	75.7	68.5	81.4	70.8	70.7	44.2

(b) Zero-shot domain results on STARv2.

Table A.1: Complete results on STARv2.

probability, and converted to a probability for NAP on all system actions a according to the policy graph edges:

$$g(i, a) = \begin{cases} p_i, & \text{if } \textbf{action}(\textbf{next}(u_i)) = a \\ 0, & \text{otherwise} \end{cases}$$

$$P(a) = \sum_{i \leq |S|} g(i, a)$$

Here, $\textbf{action}(\textbf{next}(u_i))$ gives the next system action of the user action u_i according to the policy graph. Note that p_i is an attention weight that is interepted to be the probability of user action u_i being active at the current turn; however, no supervised training was done with ground truth user action labels. Then, to implement supervised training on these user actions, we train p_i to be actual probabilities, and apply a sigmoid on p_i to form a user action prediction head. Note that this is a multilabel binary prediction. We then calculate a binary cross-entropy loss on this head.

A.6 MultiWOZ Zero-Shot Policy Program

Figure A.1 contains the ANYTOD policy program used when evaluating over MultiWOZ. This policy program was handcrafted, and provides a simplified conversation flow.

```python
def multiwoz_policy(active_domain, belief_state, act_hist):
  rec = []
  last_useracts = act_hist[-1]

  # We define a new action within the MultiWOZ schema that tracks whether
  # the user wants to book a provided entity.
  # Since this is zero-shot we don't train on this action at all, just provide
  # a natural language description "user is saying they wants to book this hotel"
  user_wants_to_book = any(act == 'user-wants-to-book' for act, _ in last_useracts)

  if user_wants_to_book:
    rec.append(('book', None))
    # Inform the name of what we're booking for the user
    if active_domain in ['restaurant', 'hotel', 'attraction']:
      rec.append(('inform', f'{active_domain}-name'))
    elif active_domain == 'train':
      rec.append(('inform', f'train-trainid'))
    # Ask the user if they need anything else
    rec.append(('reqmore', None))
  else:
    # We're still trying to find an entity for the user
    # Recommend / select entities
    if active_domain in ['restaurant', 'hotel', 'attraction']:
      rec.append(('inform', f'{active_domain}-name'))
    elif active_domain == 'train':
      rec.append(('inform', f'train-trainid'))
    rec.append(('recommend', None))
    rec.append(('select', None))
    rec.append(('booking-inform', None))

  for act, slot in last_useracts:
    if act == 'inform':
      # We often repeat back info the user has given us in next turn
      rec.append(('inform', slot))
    # If the user is requesting a slot, provide the value
    if act == 'request':
      rec.append(('inform', slot))
    # If the user is thanking us, say you're welcome / bye / anything else?
    if act == 'thank':
      rec.append(('welcome', None))
      rec.append(('bye', None))
      rec.append(('reqmore', None))

  return set(rec)
```

Figure A.1: The ANYTOD program implementation for the zero-shot policy program.

```python
USER_CUSTOM_LABEL = 'user_custom'
OUT_OF_SCOPE_LABEL = 'out_of_scope'

def anytod_star_policy_program(
    belief_state: dict[str, str], act_hist: list[list[str]], api: Json,
    graph: Json, convo_hist: list[str], primary_item: Json):
  # a list of next action prdictions to recommend to the lm
  next_act_recs = []
  # get the "bye" actions for both user and system
  user_bye_act = _user_bye_act(graph)
  sys_bye_act = _sys_bye_act(graph)

  # dict of param -> action user would take to inform this param
  slot_actions = graph['slot_actions']
  # generate a list of all user informing acts
  inform_user_acts = set()
  for _, user_acts in slot_actions.items():
    inform_user_acts.add(user_acts[0])

  if act_hist:
    # iterate through last turn's active user actions, result of AST
    for last_useract in act_hist[-1]:
      # some transitions are common to all star graphs, but not explicit
      # if user is performing something out-of-scope, return out_of_scope
      if last_useract == USER_CUSTOM_LABEL:
        next_act_recs.append(OUT_OF_SCOPE_LABEL)
      # if user is saying bye, agent can say bye
      if last_useract == user_bye_act:
        next_act_recs.append(sys_bye_act)

      # if the user is performing an action that isn't informing a param,
      # look it up in the policy graph
      if last_useract not in inform_useracts and last_useract in graph['graph']:
        next_act_recs.append(graph['graph'][last_useract])
  # if the agent can do the anything_else action, it can also say bye
  if 'anything_else' in next_act_recs:
    next_act_recs.append(bye_act)

  # if all required params are provided, we can query api
  query_label = 'query' if 'query' in graph['replies'] else 'query_check'
  if all(p.name in belief_state for p in api.params if p.required):
    next_act_recs.append(query_label)

  # param name -> api param json
  api_params_by_name = {}
  for param in api['input']:
    if param['Name'] != 'RequestType':
      api_params_by_name[param['Name']] = param
  # if a param is not known, we can request it from the user
  for slot in graph['slot_actions']:
    p = api_params_by_name[slot]
    if p.name not in belief_state:
      ask_sysact = slot_actions[p.name][0]
      next_act_recs.append(ask_sysact)

  return next_act_recs
```

Figure A.2: The ANYTOD program implementation for a given STAR policy graph.

```python
def anytod_star_trivia_policy(
    belief_state: dict[str, str], act_hist: list[list[str]], api: Json,
    graph: Json, convo_hist: list[str], primary_item: Json):
  if act_hist and len(convo_hist) >= 2:
    for last_useract in act_hist[-1]:
      # if the user is answering a question
      if last_useract == 'user_trivia_answer':
        # check that the correct trivia answer is in the user's utterance
        answer = primary_item.get('Answer', None)
        if answer:
          last_user_utt = convo_hist[-2]
          if answer.lower() in last_user_utt.lower():
            return ['trivia_inform_answer_correct_ask_next']
          else:
            return ['trivia_inform_answer_incorrect_ask_next']
  return normal_policy(belief_state, act_hist, api, graph, convo_hist,
                       primary_item)


def anytod_star_bank_policy(
    belief_state: dict[str, str], act_hist: list[list[str]], api: Json,
    graph: Json, convo_hist: list[str], primary_item: Json):
  # next_act_recs should be populated already by graph following
  # same as normal_policy() ...

  # params required for authenticating first and second way
  first_auth_slots = ['FullName', 'AccountNumber', 'PIN']
  second_auth_slots = [
      'FullName', 'DateOfBirth', 'SecurityAnswer1', 'SecurityAnswer2'
  ]
  # if either params are satisfied we can query api
  if (all(slot in bs for slot in first_auth_slots) or
      all(slot in bs for slot in second_auth_slots)):
    next_action_recs.append('query')

  # get all seen user acts
  seen_useracts = set()
  for turn, turn_acts in enumerate(act_hist):
    if turn % 2 == 0:
      seen_useracts.update(turn_acts)
  forgot_acts = ['user_bank_forgot_account_number', 'user_bank_forgot_pin']
  # if the user has forgotten anything from first auth, follow second auth
  is_second_auth = any(fa in seen_useracts for fa in forgot_acts)
  slots = second_auth_slots if is_second_auth else first_auth_slots
  if graph['task'] == 'bank_fraud_report':
    slots.append('FraudReport')
  # request slots depending on 1st/2nd auth if not known
  for slot in slots:
    if slot not in belief_state:
      next_act_recs.append(graph['slot_actions'][slot][0])

  return next_act_recs
```

Figure A.3: The ANYTOD program implementation specialized for `bank` and `trivia` domains.