Grounding Description-Driven Dialogue State Tracking

CUED IIB Project Final Report

Weixuan Zhang

Trinity College, University of Cambridge

Supervised by: Prof. Bill Byrne, Alexandru Coca

May 2023

Technical Abstract

Task-Oriented Dialogue (TOD) systems such as virtual assistants assist users with well-defined tasks, *e.g.* restaurant reservation, ticket booking. Dialogue State Tracking (DST) is an important component of TOD systems, tracking user intention in a dialogue. These intentions are usually represented using a string called *intent* (*e.g.* `Find Buses`), with details of the intention represented in key-value pairs, where the key and value are called *slot* and *value* respectively (*e.g.* `date: Sunday`).

The schema-guided paradigm of DST introduced by Rastogi et al. (2019) addresses the scalability challenges of DST models, by proposing to condition the model on task-specific *schemata* along with the dialogue. These schemata contains natural language descriptions of slots and intents associated with the particular task. However, using natural languages makes the model dependent on the linguistic styles of the schema descriptions.

Coca (2023) improves the robustness to schema variation of the state-of-the-art D3ST model (Zhao et al., 2022), by using utterances from the corpus seeking information about a slot or intent as an alternative description for that slot or intent. Five such Knowledge-Seeking Turns (KSTs) are mined from the corpus manually for each slot and intent, and are included in the D3ST prompts by either random concatenation with schema descriptions (*prompt grounding*) or data augmentation as in Lee et al. (2022).

Lee et al. (2022) show that the schema robustness is related to training prompt diversity. Building upon the success of the approach of Coca (2023), to further increase the prompt diversity, we propose to randomly sample KST from the entire corpus for each slot and intent automatically. A new code infrastructure for DST research is developed, and the random sampling approach is implemented. Other contributions include extending the official MultiWOZ evaluator to support MultiWOZ 2.4 and leave-one-domain-out evaluation, fixing errors in code released by Zhao et al. (2022), and a Python library for building processing pipelines.

Performances of data augmentation and prompt grounding are evaluated and compared with Coca (2023) and Zhao et al. (2022). Zero-shot transfer abilities are also investigated, through SGD unseen-service transfer, MultiWOZ cross-domain transfer and cross-dataset transfer experiments.

We show that random sampling further improves schema robustness compared to manual KST mining. We also confirm the applicability of KST data augmentation to the MultiWOZ dataset, by decoding using alternative slot descriptions and demonstrating an improved performance than baseline D3ST. Our approach also improves the performance of D3ST on the SGD dataset, being competitive with other state-of-the-art models. The increase is attributed to better zero-shot transfer abilities to unseen services, through exploiting the uniformity in the SGD dataset by models with higher schema robustness. Regression is observed on MultiWOZ with KST augmentation, likely due to noisy annotations. The improvement to zero-shot ability on the SGD dataset is not observed for MultiWOZ cross-domain evaluations due to the diverse and inconsistent schemata and dialogues between domains.

Contents

Acronyms **2**

1 Introduction **3**
1.1 Task-Oriented Dialogue System and Dialogue State Tracking 3
1.2 Schema-Guided Paradigm and Description-Driven DST 4
1.3 Motivation and Contribution 4

2 Background and Prior Work **5**
2.1 Datasets 5
2.1.1 Schema-Guided Dialogue (SGD) Dataset 5
2.1.2 SGD-X Dataset 5
2.1.3 MultiWOZ Datasets 5
2.2 Metrics 6
2.3 Prior Work 6

3 Experimental Methods and Infrastructure **8**
3.1 Random KST Sampling 8
3.2 Incorporation of KSTs 9
3.3 DST System 10
3.3.1 Tokeniser 11
3.3.2 Parser 11
3.3.3 Metrics Calculators 11

4 Experiments **12**
4.1 SGD 12
4.1.1 SGD Baseline and Data Augmentation 13
4.1.2 SGD Prompt Grounding 13
4.2 MultiWOZ 14
4.2.1 MultiWOZ Baseline and Data Augmentation 14
4.2.2 MultiWOZ Cross-Domain Transfer 15
4.2.3 MultiWOZ Unseen Description Transfer 15
4.3 Cross Dataset Transfer 15

5 Results and Discussion **16**
5.1 Baselines and Data Augmentation 16
5.1.1 SGD Baseline and Data Augmentation 16
5.1.2 MultiWOZ Baseline and Data Augmentation 18
5.2 Prompt Grounding and Random KST Sampling 20
5.2.1 Prompt Grounding 20
5.2.2 Grounded Decoding 21
5.3 Cross-Dataset Transfer 22

6 Conclusion **23**
6.1 Limitations and Future Work 23

References **24**

Appendix A Implementation Details **27**
A.1 Data Preprocessing . . . 27
A.1.1 Dataset-Agnostic Data Format . . . 27
A.1.2 SGD Preprocessing . . . 28
A.1.3 MultiWOZ Preprocessing . . . 29
A.2 Parser . . . 29
A.3 Runtime Preprocessor and Random KST Sampling . . . 30
A.3.1 Processing Pipeline . . . 30
A.3.2 Efficient Random KST Sampling . . . 31
A.4 MultiWOZ Evaluator . . . 32

Appendix B Errors in Code Released by Zhao et al. **34**
B.1 SGD Multi-Frame Turns Error . . . 34
B.2 MultiWOZ Block Domain Error . . . 34

Appendix C MultiWOZ Cross Domain Transfer Prompting Strategies **35**

Appendix D Other DST Models **36**

Acronyms

DAG Directed Acyclic Graph 30

DDP Distributed Data Parallel 10, 12

DST Dialogue State Tracking i, 1, 3–6, 8, 10, 17, 23, 32

JGA Joint Goal Accuracy 6, 11–13, 16–19, 21–23, 28, 29, 32, 33

KST Knowledge-Seeking Turn i, ii, 3–5, 7–10, 12–23, 27, 28, 32

NLG Natural Language Generation 3

NLU Natural Language Understanding 3

POL Dialogue Policy 3

SS Schema Sensitivity 6, 16, 17, 20–22

TOD Task-Oriented Dialogue i, 3–5

1 Introduction

In this section, we present a brief summary of dialogue state tracking in task-oriented dialogue systems and its challenges to motivate this project.

1.1 Task-Oriented Dialogue System and Dialogue State Tracking

Task-Oriented Dialogue (TOD) systems assist users with well-defined tasks, such as restaurant reservation and ticket booking (Balaraman, Sheikhalishahi, and Magnini, 2021), by providing a natural language interface to the corresponding services on the web (Rastogi et al., 2019). Due to the complexity of such systems, they often use a pipeline-based approach, containing Natural Language Understanding (NLU), Dialogue State Tracking (DST), Dialogue Policy (POL) and Natural Language Generation (NLG) (Chen et al., 2017), as illustrated in Figure 1.

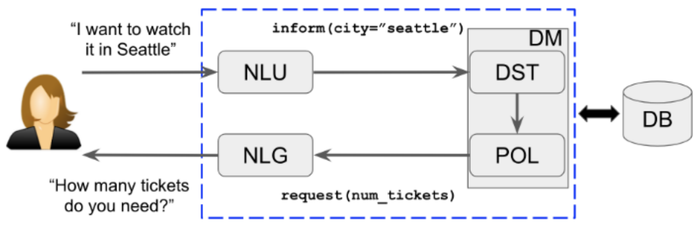
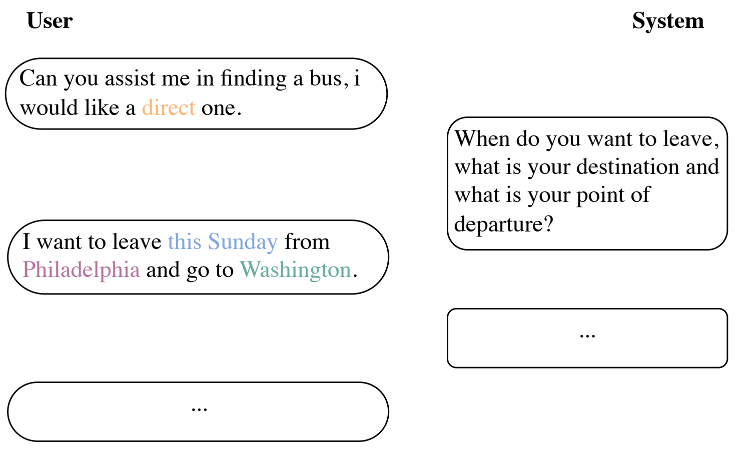

"I want to watch it in Seattle"
inform(city="seattle")
DM
NLU
DST
POL
NLG
DB
"How many tickets do you need?"
request(num_tickets)

Figure 1: Architecture of a TOD system (Gao, Galley, and Li, 2019)

DST is a key component, it identifies and tracks key information – the *dialogue state* – during a conversation, commonly represented as a sequence of *intents*, and *slot* and *value* pairs. An example dialogue about finding buses is shown in Figure 2, where the key information such as time of departure and destination is highlighted. The downstream Dialogue Policy determines the next action based on the current dialogue state, and the predicted actions are then used to generate a natural language response by NLG. Therefore, the accuracy and reliability of KST are critical to the overall performance of TOD systems.



Can you assist me in finding a bus, i would like a direct one.

When do you want to leave, what is your destination and what is your point of departure?

I want to leave this Sunday from Philadelphia and go to Washington.

...

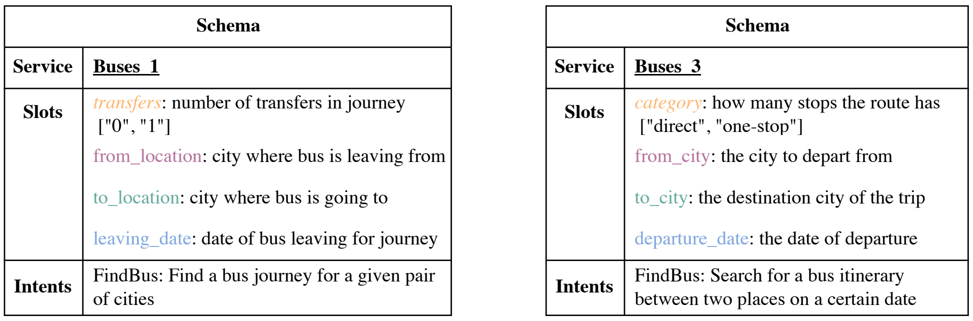
...

Figure 2: An example dialogue between a user and a TOD system. Tracked information are coloured

1.2 Schema-Guided Paradigm and Description-Driven DST

Traditionally, DST is treated as a multi-class classification problem based on fixed ontologies (Jacqmin, Rojas Barahona, and Favre, 2022), which specify all the possible values of the slots. However, this has scalability issues – a large amount of data needs to be collected to train a new model every time a new service is incorporated. This is not feasible for real-world TOD systems, which need to support a constantly increasing number of services.

Rastogi et al. (2019) propose the schema-guided paradigm, where each service is defined using a *schema* that contains the natural language descriptions of all slots and intents of that service. These schemata can be explicitly used as an input to the DST model, potentially enabling a unified model to support all services. Unlike the approaches using fixed ontologies, the schema slots can be "open", *i.e.* with no constraint on the possible values. On the other hand, slots can also be defined to be *categorical*, where the possible values are defined. Examples of the schemata corresponding to services of the domain in Figure 2 are shown in Figure 3.



Schema	
Service	**Buses_1**
Slots	*transfers*: number of transfers in journey ["0", "1"]
	from_location: city where bus is leaving from
	to_location: city where bus is going to
	leaving_date: date of bus leaving for journey
Intents	FindBus: Find a bus journey for a given pair of cities

Schema	
Service	**Buses_3**
Slots	*category*: how many stops the route has ["direct", "one-stop"]
	from_city: the city to depart from
	to_city: the destination city of the trip
	departure_date: the date of departure
Intents	FindBus: Search for a bus itinerary between two places on a certain date

Figure 3: Examples of schema from the domain of dialogue in Figure 2. The corresponding slots are coloured accordingly. *Italic* slots are categorical

Recent advances in DST rely on pre-trained language models (Jacqmin, Rojas Barahona, and Favre, 2022). D3ST (Zhao et al., 2022) is an example, using a fine-tuned T5 (Raffel et al., 2020) model and the schema-guided approach. It uses the slots descriptions from the schemata (the *prompt*) and the dialogue context as the encoder input and predicts the dialogue states in a sequence-to-sequence fashion.

1.3 Motivation and Contribution

D3ST achieves state-of-the-art performance, but using the natural language schema descriptions makes it sensitive to the writing style of the schemata (Lee et al., 2022). Since the schemata are often written by service developers, their writing style is not consistent, so the robustness to schema variation is important.

Lee et al. (2022) demonstrate the robustness increases when the training data is augmented with paraphrases of the descriptions. However, the augmentation data is often expensive to collect. Coca (2023) propose to augment the training data of D3ST with five Knowledge-Seeking Turns (KSTs) mined manually from the dialogue corpus itself. They also propose to ground the D3ST prompts using KSTs, achieving significant improvements to model schema robustness.

As prompt diversity is important for robustness (Lee et al., 2022), instead of using five hand-picked turns as Coca (2023), KSTs can be randomly sampled from the corpus automatically. In this project, an extensible code infrastructure for DST research is developed, and the random KST sampling approach is implemented and evaluated.

2 Background and Prior Work

The datasets and evaluation metrics used in this project are discussed in this section. The prior work this project build upon is also presented.

2.1 Datasets

2.1.1 Schema-Guided Dialogue (SGD) Dataset

The SGD dataset (Rastogi et al., 2019) is the largest public TOD corpus, containing more than 16000 dialogues over 16 domains in the training set. The dataset introduces the schema-guided paradigm, where each domain consists of a number of services, each defined by a schema. The schema describes the slots and intents of a service, and mirrors the information required to query real-world services. The test set contains unseen services not in the training set, reflecting the challenge of incorporating new services in the real world.

The dataset is constructed using the machine-machine interaction approach (Shah et al., 2018), where the dialogue outlines are first generated through interactions of two agents using probabilistic automata based on an initial scenario. The dialogue outlines consist of *dialogue acts* that may contain *act parameters* (*e.g.* `REQUEST(location)` denotes an utterance aiming to request information about the slot `location`), and controls the flow of the dialogue. Then natural language dialogues are paraphrased from the outlines using crowd-workers, with help from a pre-defined mapping between dialogue acts and utterances.

2.1.2 SGD-X Dataset

The SGD-X dataset (Lee et al., 2022) contains five variants of every schema in the SGD dataset, with names and descriptions replaced with semantically similar paraphrases. It enables the evaluation of model robustness to schema variations. The variants are of increasing diversity, with variant `v1` the most similar to the SGD schema.

2.1.3 MultiWOZ Datasets

The MultiWOZ dataset (Budzianowski et al., 2018) contains over 10k dialogues collected via a Wizard-of-Oz (WOZ) setup (Kelley, 1984), where each dialogue is constructed by two crowd-workers, who also annotate the dialogue states, communicating with each other. The corpus consists of 7 domains, with about 67% of the dialogues containing multiple domains. Each domain is defined by an ontology, which contains the slots and all possible values.

Due to the data collection set-up, the original MultiWOZ dataset contains numerous annotation errors and inconsistencies, four further versions are later released aiming to improve the annotation quality.

MultiWOZ 2.1 (Eric et al., 2020). MultiWOZ 2.1 uses crowd-workers to re-annotate the dialogue state annotations, and consolidates follow-up work to include user dialogue acts and multiple slot descriptions for each slot.

MultiWOZ 2.2 (Zang et al., 2020). MultiWOZ 2.2 fixes more state annotations on top of MultiWOZ 2.1. It follows the schema-based approach by redefining the ontology into schemata similar to those in the SGD dataset. Categorical and non-categorical slots are classified by the number of values in the MultiWOZ 2.1 ontology. Some turns in MultiWOZ 2.1 do not have dialogue act annotations, they are added using crowdsourcing with slot names modified to match the new schema.

MultiWOZ 2.3 (Han et al., 2020). MultiWOZ 2.3 improve upon MultiWOZ 2.1, focusing on dialogue acts. It implements co-reference features and unifies dialogue act and state annotations. However, the dialogue act annotations are not consistent with the MultiWOZ 2.2 schema.

MultiWOZ 2.4 (Ye, Manotumruksa, and Yilmaz, 2022). MutliWOZ 2.4 improves the annotations in the validation and test sets, on top of MultiWOZ 2.1, to improve the correctness of model evaluation. The train set is unchanged from MultiWOZ 2.1.

2.2 Metrics

Two metrics are used to evaluate the performance of DST.

Joint Goal Accuracy (JGA). The average accuracy of predicting all slot assignments for a turn correctly. The SGD-X dataset allows the calculation of average JGA across the schema variants, denoted here as $JGA_{v_{1-5}}$.

Schema Sensitivity (SS). The SGD-X dataset introduces Schema Sensitivity of JGA – the turn-level coefficient of variation of JGA across the schema variants, as defined by Lee et al. (2022). It measures the consistency of model predictions across schema variants – a model with lower SS is more robust to schema variations.

A model robust to schema variations should have a high $JGA_{v_{1-5}}$ and a low SS, since if a model consistently omits slots, it will also have a low SS.

2.3 Prior Work

D3ST. D3ST is a state-of-the-art description-driven DST model proposed by Zhao et al. (2022).

Since the naming convention for slots and intents is often not consistent in schema-guided approaches, and the names may convey little semantic meaning (Zhao et al., 2022), models may memorise patterns in data. Therefore, Zhao et al. (2022) replace names and notations with natural language descriptions from the corresponding service schema, with an index-picking mechanism, as shown in Figure 4.

Each slot is assigned a random integer index between 0 and the number of slots in the service schema (exclusive). The random assignment aims to prevent the model from memorising the order of index-description pairs. For categorical slots (*e.g.* slot `category` in Figure 4), the possible values are added to the slot description, prefixed with the slot index and a letter (as shown by the purple substring). Similarly, each intent is also assigned an index randomly, but with the prefix "`i`" before the integers.

The slot/intent indices and descriptions are joined together with the "`=`" delimiter, forming the *prompt*. The prompt is prefixed to the *dialogue context* and used as the inputs to a sequence-to-sequence (seq2seq) model, which is trained to predict the target – a formatted string of dialogue states. The dialogue context is the utterances of all turns of the dialogue up to the current turn joined together, with the user utterances and system utterances prefixed by "`[user]`" and "`[system]`" respectively. The target contains slot indices and corresponding slot values (with categorical values specified using their indices), and intent indices.

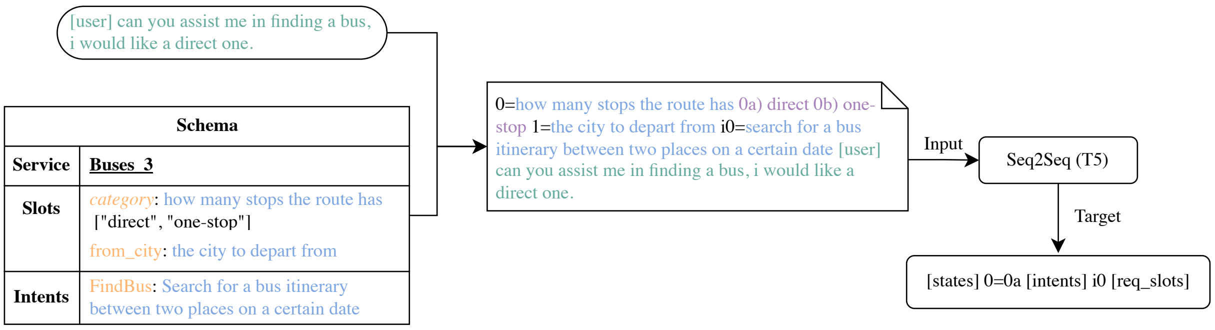

[user] can you assist me in finding a bus, i would like a direct one.
Schema
Service
Buses_3
Slots
category: how many stops the route has ["direct", "one-stop"]
from_city: the city to depart from
Intents
FindBus: Search for a bus itinerary between two places on a certain date
0=how many stops the route has 0a) direct 0b) one-stop 1=the city to depart from i0=search for a bus itinerary between two places on a certain date [user] can you assist me in finding a bus, i would like a direct one.
Input
Seq2Seq (T5)
Target
[states] 0=0a [intents] i0 [req_slots]

Figure 4: An example D3ST formatted input-target pair. Dialogue context in green, schema descriptions in blue, possible categorical values in purple

Grounding D3ST. Coca (2023) propose using the Knowledge-Seeking Turns as alternative schema descriptions. Utterances annotated with the `REQUEST` dialogue act (defined in Section 2.1.1) can be used as alternative slot descriptions. Similarly, turns annotated with `INFORM_INTENT` or `OFFER_INTENT` can be used for intents. Coca (2023) select five turns with a single such action annotation from the corpus randomly manually. For slots that do not have KST or have less than five KSTs, the turns are chosen from the KSTs of other services if the slot also exists in those services. Otherwise, the relevant span appearing before or after the slot value is selected from utterances annotated with the `INFORM` or `CONFIRM` dialogue actions with action parameters equalling the slot-value pair. This is called *span selection*.

The mined KSTs are incorporated into D3ST formatted prompts following three different methods, as shown in Figure 5. In the `Turn` prompt format, the selected KST is concatenated to the original schema description in a random order, separated by the delimiter "`;`". The random concatenation is to prevent the model from attending to one of the two descriptions (Coca, 2023). The categorical slot value candidates and their indices are always at the end of the concatenated string (as shown by the purple substrings in Figure 5). The `TurnSlot` format also includes the slot names during the random concatenation, to exploit information contained in the slots names. The names are transformed into forms of natural language (*e.g.* "`from_city`" is changed to "`from city`") to be consistent with the other two sources – schema description and KST. Both `Turn` and `TurnSlot` form a new dataset that is as large as the SGD dataset, and *grounds* the prompts.

Other than grounding the prompts by incorporating KSTs on a per-slot/intent basis, the original prompt can be *augmented* with five variants, which are formed by replacing the original prompt with the mined KSTs entirely. This prompt format is called `KST-DA`, and forms a new dataset six times as large as the original (*i.e.* as large as SGD-X).

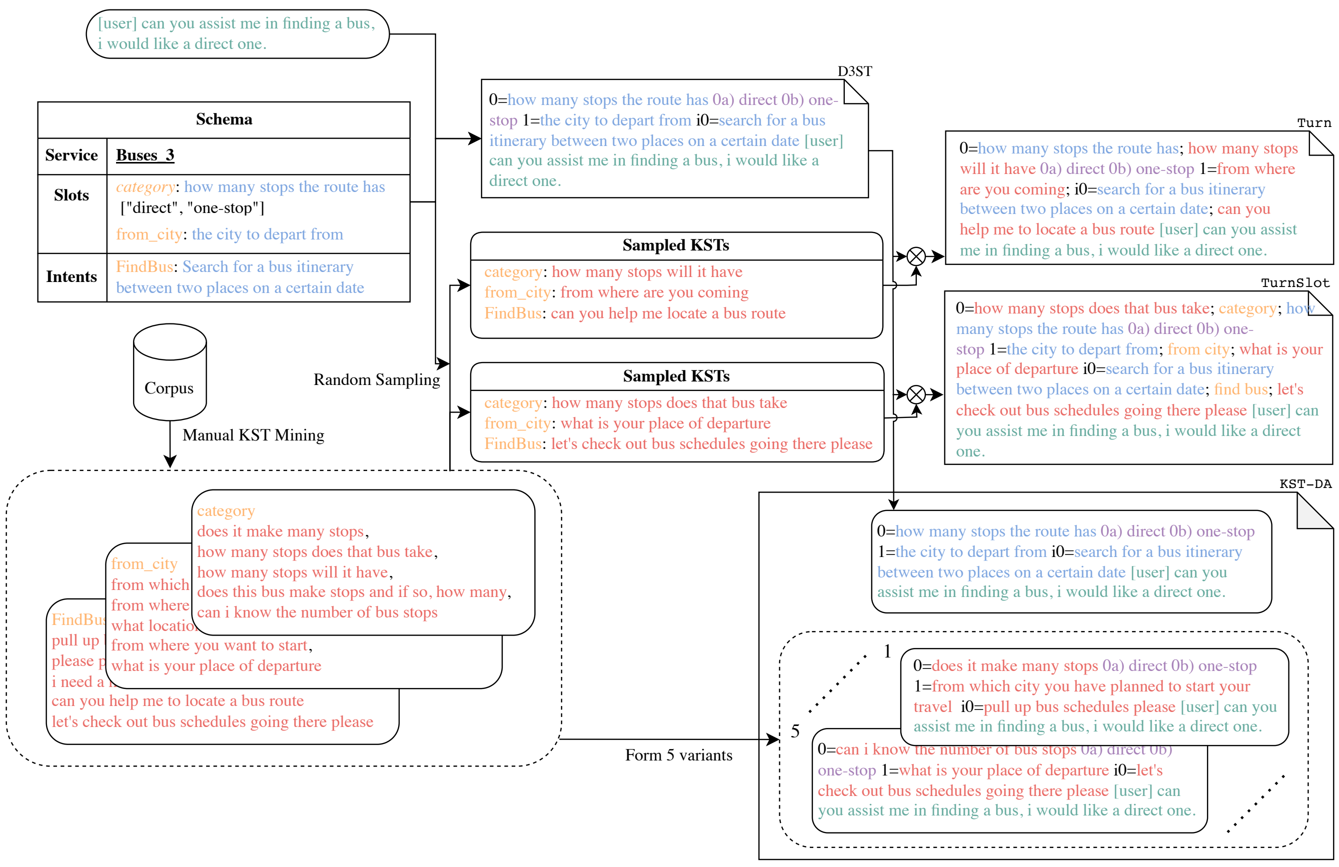

[user] can you assist me in finding a bus, i would like a direct one.
Schema
Service
Buses_3
Slots
category: how many stops the route has ["direct", "one-stop"]
from_city: the city to depart from
Intents
FindBus: Search for a bus itinerary between two places on a certain date
Corpus
Manual KST Mining
Random Sampling
D3ST
0=how many stops the route has 0a) direct 0b) one-stop 1=the city to depart from i0=search for a bus itinerary between two places on a certain date [user] can you assist me in finding a bus, i would like a direct one.
Sampled KSTs
category: how many stops will it have
from_city: from where are you coming
FindBus: can you help me locate a bus route
Sampled KSTs
category: how many stops does that bus take
from_city: what is your place of departure
FindBus: let's check out bus schedules going there please
Turn
0=how many stops the route has; how many stops will it have 0a) direct 0b) one-stop 1=from where are you coming; i0=search for a bus itinerary between two places on a certain date; can you help me to locate a bus route [user] can you assist me in finding a bus, i would like a direct one.
TurnSlot
0=how many stops does that bus take; category; how many stops the route has 0a) direct 0b) one-stop 1=the city to depart from; from city; what is your place of departure i0=search for a bus itinerary between two places on a certain date; find bus; let's check out bus schedules going there please [user] can you assist me in finding a bus, i would like a direct one.
KST-DA
0=how many stops the route has 0a) direct 0b) one-stop 1=the city to depart from i0=search for a bus itinerary between two places on a certain date [user] can you assist me in finding a bus, i would like a direct one.
1
0=does it make many stops 0a) direct 0b) one-stop 1=from which city you have planned to start your travel i0=pull up bus schedules please [user] can you assist me in finding a bus, i would like a direct one.
5
0=can i know the number of bus stops 0a) direct 0b) one-stop 1=what is your place of departure i0=let's check out bus schedules going there please [user] can you assist me in finding a bus, i would like a direct one.
Form 5 variants
category
does it make many stops,
how many stops does that bus take,
how many stops will it have,
does this bus make stops and if so, how many,
can i know the number of bus stops
from_city
from which
from where
what locatio
from where you want to start,
what is your place of departure
FindBu
pull up
please p
i need a
can you help me to locate a bus route
let's check out bus schedules going there please

Figure 5: Methods of incorporating mined KSTs into D3ST formatted prompts. Dialogue context in green, schema descriptions in blue, sampled KST in red, possible categorical values in purple, slot names in orange

3 Experimental Methods and Infrastructure

A large portion of this project is devoted to the engineering of a reliable and extensible code infrastructure for DST research, enabling random KST sampling, and developing D3ST implementation and evaluator for the MultiWOZ dataset, which is non-trivial and requires significant engineering efforts.

We call the new system Robust-DST, whose implementation including open-source contributions are discussed in detail in Appendix A. The high-level system design and methods are discussed in this section. The code base is tested extensively, discovering mistakes in code released by Zhao et al. (2022) in the process. The errors and their potential effects are discussed in Appendix B.

The datasets are first preprocessed into a dataset-agnostic data format (Section A.1.1), which contains model inputs and targets in the D3ST format, and other metadata required by the DST system.

3.1 Random KST Sampling

The random sampling of KSTs is done by introducing a runtime preprocessor, which further processes the preprocessed datasets before tokenisation. The flow and transformation of data are shown in Figure 6, where each rectangular block corresponds to a processing step.

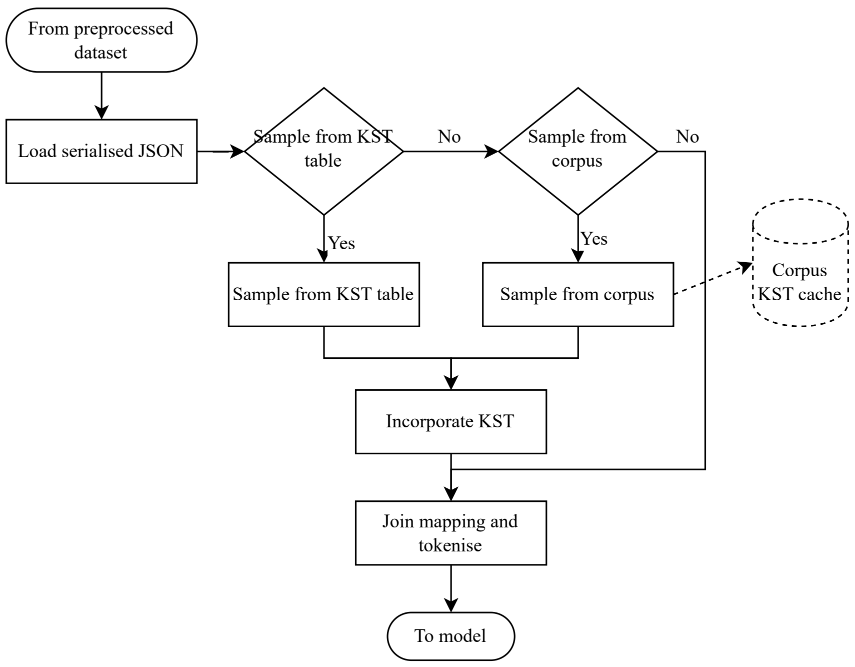


Figure 6: Runtime preprocessing pipeline data flow and transformations

KSTs can be sampled from a table containing valid turns for each slot and intent, or from the corpus automatically.

When sampling from the corpus, the dataset is iterated twice. During the first iteration, a cache containing valid KSTs to sample from for a particular slot or intent is built, using the action annotations. Some turns have more than one action or one action acting on several slots, with corresponding utterances often having multiple sentences or questions. Since there is no clear way of extracting parts of the utterance targeting the desired slot only, turns with a single knowledge-seeking action acting on a single slot or intent are used. Furthermore, some turns annotated with the `REQUEST` action are confirmations, *e.g.* `whether are you travelling to 351 west washington avenue`, and are not included. There are a total of 3930 such conformational turns in the SGD training set, out of 34774 turns selected. The cache is used during the second iteration to randomly select a corresponding KST for each slot and intent that have such turns. Since our approach does not use KSTs from other services or perform span selection (which is very difficult to automate), as used by Coca (2023) when mining the turns manually, some slots do not have KSTs to sample from.

The sampled KST are then incorporated into the original D3ST formatted prompt before tokenisation.

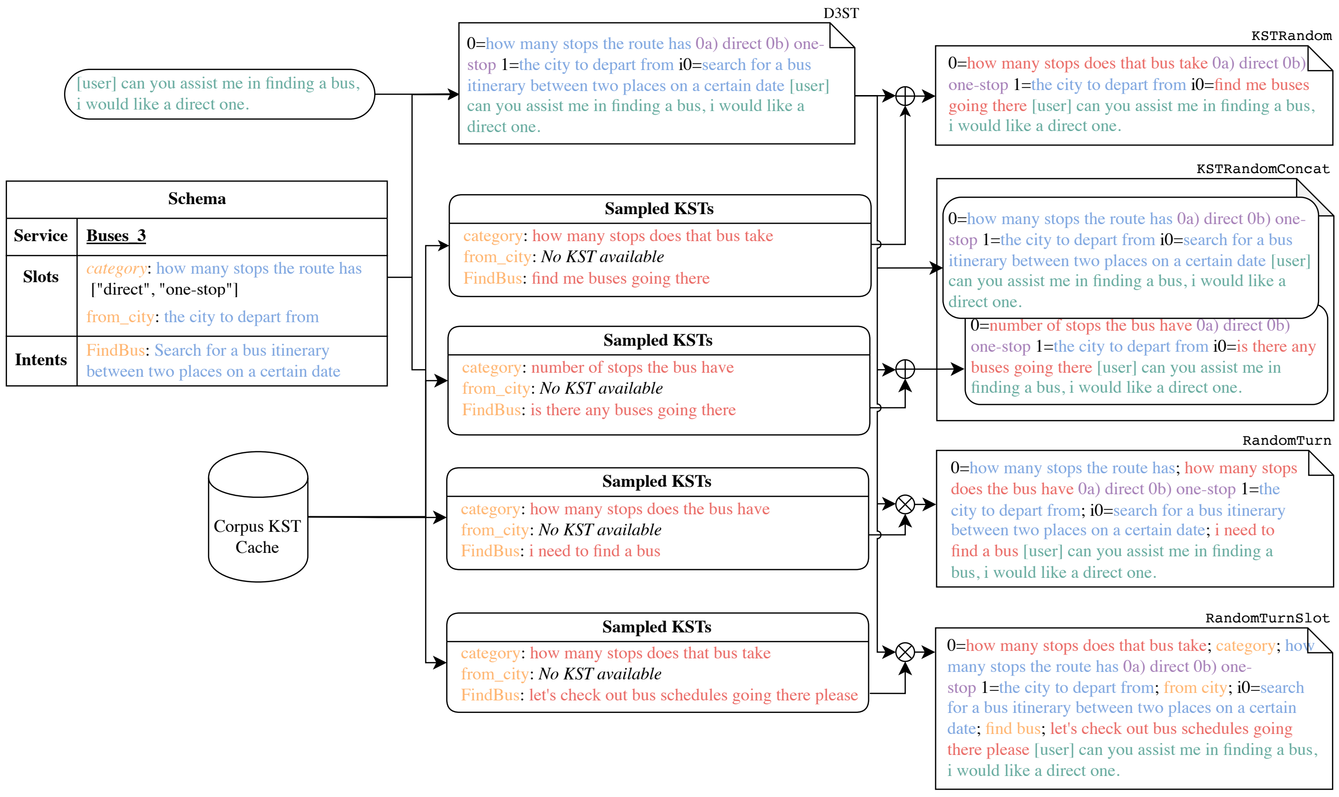
3.2 Incorporation of KSTs

The randomly sampled KSTs can be incorporated into the D3ST formatted prompts in several ways, as shown in Figure 7.

Similar to when the KSTs are mined manually in Figure 5, the prompts can be grounded by random concatenation of schema descriptions and KSTs, defined here as `RandomTurn`. When the slot and intent names are also included, we call the prompt format `RandomTurnSlot`.

On the other hand, as random sampling leads to a greater number of KSTs for a given dialogue, all slot and intent descriptions of the original D3ST prompt can be replaced with sampled KSTs without affecting the prompt diversity. This is named `KSTRandom`, the new dataset is of the same size as the SGD dataset.

`KSTRandomConcat` is analogous to `KST-DA`, where the dataset is augmented by a `KSTRandom` formatted prompt for each D3ST formatted prompt in the dataset, *i.e.* a concatenation between the D3ST processed dataset and a transformed dataset following the `KSTRandom` prompt format. The resulting dataset is twice as large as the original.



[user] can you assist me in finding a bus, i would like a direct one.
Schema
Service
Buses_3
Slots
category: how many stops the route has ["direct", "one-stop"]
from_city: the city to depart from
Intents
FindBus: Search for a bus itinerary between two places on a certain date
Corpus KST Cache
D3ST
0=how many stops the route has 0a) direct 0b) one-stop 1=the city to depart from i0=search for a bus itinerary between two places on a certain date [user] can you assist me in finding a bus, i would like a direct one.
Sampled KSTs
category: how many stops does that bus take
from_city: No KST available
FindBus: find me buses going there
Sampled KSTs
category: number of stops the bus have
from_city: No KST available
FindBus: is there any buses going there
Sampled KSTs
category: how many stops does the bus have
from_city: No KST available
FindBus: i need to find a bus
Sampled KSTs
category: how many stops does that bus take
from_city: No KST available
FindBus: let's check out bus schedules going there please
KSTRandom
0=how many stops does that bus take 0a) direct 0b) one-stop 1=the city to depart from i0=find me buses going there [user] can you assist me in finding a bus, i would like a direct one.
KSTRandomConcat
0=how many stops the route has 0a) direct 0b) one-stop 1=the city to depart from i0=search for a bus itinerary between two places on a certain date [user] can you assist me in finding a bus, i would like a direct one.
0=number of stops the bus have 0a) direct 0b) one-stop 1=the city to depart from i0=is there any buses going there [user] can you assist me in finding a bus, i would like a direct one.
RandomTurn
0=how many stops the route has; how many stops does the bus have 0a) direct 0b) one-stop 1=the city to depart from; i0=search for a bus itinerary between two places on a certain date; i need to find a bus [user] can you assist me in finding a bus, i would like a direct one.
RandomTurnSlot
0=how many stops does that bus take; category; how many stops the route has 0a) direct 0b) one-stop 1=the city to depart from; from city; i0=search for a bus itinerary between two places on a certain date; find bus; let's check out bus schedules going there please [user] can you assist me in finding a bus, i would like a direct one.

Figure 7: Methods of incorporating randomly sampled KSTs into D3ST formatted prompts. Assume slot `from_city` does not have any KSTs

If a slot does not have KSTs, the original schema description is kept for `KSTRandom` and `KSTRandomConcat`, as shown by the blue substrings for slot `1` in Figure 7. For `RandomTurn` and `RandomTurnSlot`, no KSTs are concatenated, but the delimiter "`;`" is still appended for consistency.

3.3 DST System

The DST system consists of six main components – the runtime data preprocessor mentioned earlier in Section 3.1, tokeniser, data loader and model trainer, sequence-to-sequence model, output parser, and metrics calculators.

The model tokeniser, data loader, model and model trainer are developed using the *HuggingFace* library (Wolf et al., 2019). It supports multi-GPU training using PyTorch Distributed Data Parallel (DDP), which is important due to the memory requirement to fine-tune the model. Similar to Coca (2023) and Zhao et al. (2022), the T5 transformer (Raffel et al., 2020) is used as the sequence-to-sequence model.

The parser and metrics calculators are used during model evaluation, where the operations are shown in Figure 8.

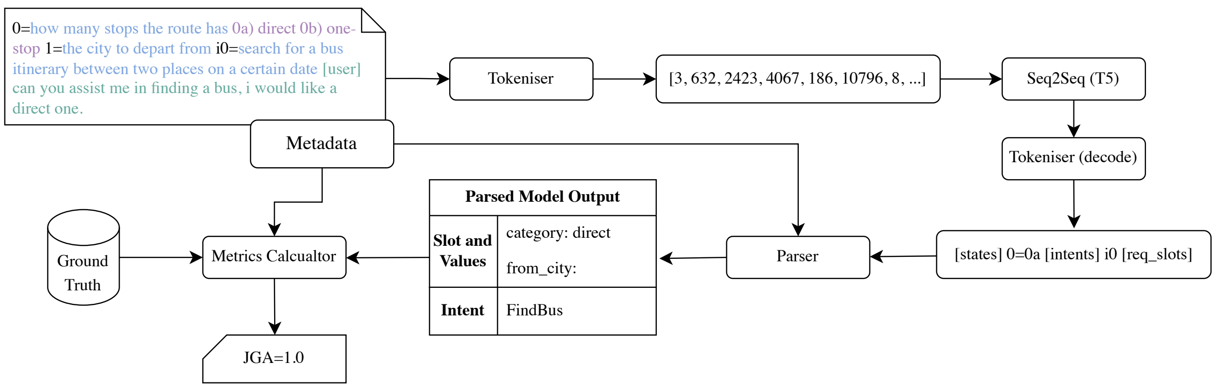

0=how many stops the route has 0a) direct 0b) one-stop 1=the city to depart from i0=search for a bus itinerary between two places on a certain date [user] can you assist me in finding a bus, i would like a direct one.
Tokeniser
[3, 632, 2423, 4067, 186, 10796, 8, ...]
Seq2Seq (T5)
Tokeniser (decode)
[states] 0=0a [intents] i0 [req_slots]
Parser
Metadata
Parsed Model Output
Slot and Values
category: direct
from_city:
Intent
FindBus
Ground Truth
Metrics Calcualtor
JGA=1.0

Figure 8: Operations during evaluation

3.3.1 Tokeniser

The pretrained T5 tokeniser is used. Special delimiters in the D3ST prompts such as `[states]`, `[intents]` are added as special tokens to the tokeniser. Otherwise, much more tokens are produced by the tokeniser, making the inputs more likely to reach the maximum input length limit and get truncated, and also making it more difficult for the model to learn the output syntactic structure.

3.3.2 Parser

The parser parses the model output string (an example is shown in Figure 8) into a mapping from slot names to predicted values. This allows the prediction to be scored using the metric calculators.

3.3.3 Metrics Calculators

The metric calculators calculate Joint Goal Accuracy (JGA) from the parsed outputs, by comparing the slot values with the ground truth.

SGD. Metrics when using the SGD dataset are calculated using the official evaluation script (Rastogi et al., 2019), by writing the parsed outputs into blank SGD formatted dialogue files. For non-categorical slots, the predicted values are compared with the ground truth values using fuzzy matching. The JGA is calculated as the product of individual slot accuracies (0 or 1.0 for categorical slots and the fuzzy matching score for non-categorical slots). Other than the overall JGA, the JGAs of seen and unseen services are also calculated.

MultiWOZ. The official MultiWOZ evaluator (Nekvinda and Dušek, 2021) is used. However, it only supports evaluation against the MultiWOZ 2.2 dataset, it is extended to support MultiWOZ 2.4 to benefit from the improvements. The ability to evaluate in the leave-one-domain-out setting is also added, to enable cross-domain transfer evaluations. The limitation of the official evaluator is discussed in Section A.4, along with the extensions.

Under the leave-one-domain-out setting, we follow the evaluation approach taken by Campagna et al. (2020). When calculating JGA^{unseen} of a model trained leaving a domain out, only turns containing that unseen domain are considered. The turns with an empty belief state at the start of a dialogue segment about that unseen domain are also considered.

The algorithm is detailed in Listing 4. Among these turns, one is considered to be correctly predicted (and thus contributes to an increase in JGA) when all the slots from the unseen domain are predicted correctly, while the predictions of slots from the seen domain do not affect the results.

The JGA calculation logic is unchanged from the official evaluator. A turn is considered correct if all the fuzzy matching scores between the normalised values and the corresponding ground truths are above 0.95, categorical slots are not treated differently.

4 Experiments

The experiments are designed to answer the following questions:

1. Whether random KST sampling further improves model robustness to variations in schema descriptions when compared to manual KST mining with a limited number of turns, as done by Coca (2023)

2. Whether the approaches are generalisable to the MultiWOZ dataset, by comparing robustness to slot description variations with the D3ST baseline (Zhao et al., 2022)

3. Whether the approaches improve D3ST performances and facilitate zero-shot transfer to unseen tasks

In this section, details of the experiments conducted and their motivations will be discussed.

All the experiments are based on the `google/t5-v1_1-base` (250M parameters) checkpoint released on *HuggingFace*, which is fine-tuned using the Adafacotr optimiser. Following Coca (2023) and Zhao et al. (2022), an effective batch size of 32, and a constant learning rate of 1×10^{-4} with a linear warm-up schedule are used. The fine-tunings are completed on the Cambridge Research Computing Services HPC, distributed across 2 Nvidia A100 80GB GPUs on a single compute node using DDP for 15 epochs. During training, the model is evaluated on the validation set approximately every epoch, and the training is early terminated if three consecutive evaluations show no improvement to the highest JGA achieved. The model used for testing is the one with the highest evaluation JGA during training. All experiments are repeated three times with different random seeds.

4.1 SGD

The SGD dataset is first preprocessed into the D3ST format using a modified preprocessing script released by Zhao et al. (2022), the details of the modifications are discussed in Section A.1.2. The preprocessing differences are not reported in Zhao et al. (2022). We follow Zhao et al. (2022) and treat categorical slots with only numerical possible values as non-categorical[1]. The SGD experiments use input and output token lengths of 1024 and 512 respectively. The number of warm-up steps is set to 1000. The models are decoded on both the SGD and SGD-X datasets.

[1]This is not reported by Zhao et al. (2022) in their paper, and is confirmed upon examination of their released code

4.1.1 SGD Baseline and Data Augmentation

The new experimental infrastructure is first tested by reproducing the D3ST baseline on the SGD dataset. Then random sampling of KSTs is applied, and the sampled KSTs are incorporated into the prompts using the `KSTRandom` and `KSTRandomConcat` prompt formats as described in Section 3.2. The models are decoded with the original D3ST prompt format.

Each GPU is assigned a batch size of 16, with no gradient accumulation. The average times taken to fine-tune the baseline models and `KSTRandom` models are 6.1 hours and 7.3 hours respectively. The `KSTRandomConcat` experiments, with a dataset twice as large, take 12.4 hours on average.

4.1.2 SGD Prompt Grounding

After the approach and experimental infrastructure are validated through the data augmentation experiments, models are fine-tuned with the more complicated `RandomTurn` and `RandomTurnSlot` prompt formats (Section 3.2). As the prompts are longer than before, each GPU is assigned a batch size of 8 with a gradient accumulation step of 2 to avoid out-of-memory errors. The input token length is still set to 1024, but the truncated examples are discarded to prevent the model from learning the turns with truncated incomplete dialogue contexts. On average, approximately 84 and 226 examples are discarded with `RandomTurn` and `RandomTurnSlot` respectively depending on the random seed, out of the total 175780 training examples. The `RandomTurn` and `RandomTurnSlot` experiments take 8.3 and 11.2 hours on average respectively. The models are selected based on JGAs on the original SGD validation set.

During training, the KSTs are sampled from the entire training corpus. When decoding with the grounded prompts, the exact five KSTs used by Coca (2023) for each slot and intent are employed for comparability. This is achieved using the runtime preprocessor with a KST table (Section 3.1) built from the desired turns, instead of sampling from the corpus. The constructed datasets are saved and used throughout the grounded prompt decoding experiments to eliminate the effects from the random sampling of the exact KST to use for each slot/intent. The same KSTs are also used when decoding on SGD-X, for all of the five schema variants.

However, as mentioned in Sections 3.1 and 3.2, the random sampling approach does not use turns from other services or perform span selection. If a slot does not have any KSTs, nothing is concatenated to the original schema descriptions when using the `RandomTurn` prompt format, or to the randomly shuffled concatenation between descriptions and slot/intent names when using `RandomTurnSlot`. This means the seen slots in the test set (from seen services) are prompted differently during training. Out of the 215 slots in the SGD training set, there are 41 slots without or having less than five KSTs (when omitting conformational turns). Among these slots, eight slots without KSTs and three slots with less than five KSTs are present in the test set.

The seen slots in the test set are prompted in the same way as during training, to aid recall of the model. This is done by filtering the KST table and removing any turns not in the training set corpus for seen slots. The full five turns are used for sampling for the unseen slots. The `RandomTurn` and `RandomTurnSlot` models are decoded with their training prompt formats. The `KSTRandom` and `KSTRandomConcat` models from Section 4.1.1 are also decoded with the same datasets employing `RandomTurn` and `RandomTurnSlot` prompt formats.

4.2 MultiWOZ

To investigate the applicability of the random KST sampling approach to the MultiWOZ dataset, similar experiments to those on the SGD dataset are conducted on the MultiWOZ 2.4 dataset, with some files taken from other MultiWOZ versions:

- The schema is taken from MultiWOZ 2.2, which follows the schema-based approach
- Some versions of the MultiWOZ dataset include a slot description file, which contains as least two variants of natural language slot descriptions. Since MultiWOZ 2.4 does not contain a slot description file, the one from MultiWOZ 2.1 is used. We follow Zhao et al. (2022) and use the first description for each slot when constructing the D3ST prompts[2].
- The MultiWOZ 2.4 dialogue acts annotations are unmodified from MultiWOZ 2.1, which is incomplete and inconsistent with the MultiWOZ 2.2 schema slot names, as discussed in Section 2.1.3. Therefore, the dialogue acts annotations from MultiWOZ 2.2 is used instead.

As mentioned in Section 2.1.3, the categorical slots are defined as an afterthought when introducing MultiWOZ 2.2, so the candidate values for the categorical slots do not contain all possible values in the corpus. We follow Zhao et al. (2022) and replaces categorical slot value with "`unknown`", if it is not listed in the schema[3].

4.2.1 MultiWOZ Baseline and Data Augmentation

Since MultiWOZ is not constructed with schemata separated by services, it does not have the notion of intent. This means when decoding, the domains contained in the dialogue are unknown. The same approach taken by Zhao et al. (2022) is used, where the descriptions of slots from all domains in the ontology are included in the prompt, and an input length of 2048 is used. The domain names are prefixed to the slot descriptions to avoid ambiguity between descriptions from different domains, *e.g.* for slot `pricerange` from the `hotel` domain, the prefixed slot description is "`hotel-price budget of the hotel 1a) moderate 1b) cheap 1c) expensive`".

Including all slot descriptions and domain names in the prompt leads to excessively long prompts, if keep using 1024 as the input length, 45% of the 56778 turns in the MultiWOZ training set would be truncated. Using 2048 tokens ensures no truncation takes place. Due to the long prompts, `RandomTurn` and `RandomTurnSlot` are not used with MultiWOZ.

After the implementation is validated through baseline models achieving comparable performances to Zhao et al. (2022), `KSTRandom` and `KSTRandomConcat` prompt formats are used. Gradients are accumulated over two batches, with a per GPU batch size of 8. Even though the encoder inputs are longer, as the MultiWOZ dataset contains fewer examples, the average time to fine-tune the baseline models is 5.5 hours. The times required for `KSTRandom` and `KSTRandomConcat` are 7.5 and 7.3 hours respectively.

[2]This is not reported by Zhao et al. (2022) in their paper, and is confirmed upon examination of their released code

[3]Also not reported by Zhao et al. (2022)

4.2.2 MultiWOZ Cross-Domain Transfer

To assess whether training with augmented KSTs facilitates knowledge transfer when predicting slots from new domains not seen in training, the models are trained and evaluated in a cross-domain setting, which is also reported by Zhao et al. (2022). We aim to take an approach similar to TransferQA (Lin et al., 2021b) and T5DST (Lin et al., 2021a) (both are cited by Zhao et al. (2022)). However, both TransferQA and T5DST predicts one slot at a time, while D3ST predicts all the slots in the given turn together. There are several logical ways to prompt the model, while details of the prompt formats are not included in Zhao et al. (2022). Through communication with the authors of D3ST, their approach is taken, but the other logical approaches are explored in Appendix C.

Models are trained with one domain out of `attraction`, `hotel`, `restaurant`, `taxi` and `train` left-out. Note that only five of the seven domains in MultiWOZ are considered, as the test set only contains these five domains (Lin et al., 2021a). When a particular domain is left-out during training, turns containing slots from domains not among the other four are discarded. Models trained with that domain left out are then tested on the test set of that domain only, *i.e.* turns from the test set containing slots from any other domains are discarded.

Instead of including descriptions of all slots in the ontology (as in Section 4.2.1) in the prompt, only those of slots from domains that have non-empty slot values in the belief state – the active domains – are included. We call this approach using active domains only. This prompting strategy is used for both training and decoding. It is valid when decoding under this setting because the test set contains turns with slots from the unseen domain only, there is no ambiguity of domains active in a turn. The domain names are also prefixed as in Section 4.2.1.

Significant computing resources are required to train three models for each of the five domains being left out, even though there are fewer training examples and the encoder inputs are shorter. Due to the results of Section 4.2.1, only `KSTRandomConcat` is applied and its results are compared with the reproduced D3ST baselines. On average, each of the 30 models takes approximately 2.5 hours to fine-tune.

4.2.3 MultiWOZ Unseen Description Transfer

As mentioned earlier, only the first of the available slot descriptions are used in the prompts. To evaluate the robustness to description variations, a new test set is constructed using the second descriptions. This is analogous to SGD-X. Models from Sections 4.2.1 and 4.2.2 are evaluated on this new dataset.

4.3 Cross Dataset Transfer

Since the experimental infrastructure supports both SGD and MultiWOZ datasets, the applicability of models trained on one dataset to the other is investigated, to further evaluate the zero-shot abilities.

Upon private communication with the authors of Zhao et al. (2022), the MultiWOZ dataset is processed with active domains only (as in Section 4.2.2), but without filtering of turns or adding the domain names. This results in the prompts with variable lengths, and better matches prompts in the SGD dataset. Instead of the MultiWOZ models from Section 4.2.1, new models are trained with these prompts. The SGD models from Section 4.1.1 are directly applied to MultiWOZ with the new prompts.

5 Results and Discussion

We now present our experimental findings. All experiments reported are averages of three runs, with different random seeds

5.1 Baselines and Data Augmentation

5.1.1 SGD Baseline and Data Augmentation

The baseline D3ST models are reproduced with the new experimental infrastructure, the results are shown in Row 4 of Table 1. Other than the modifications mentioned in Section 4.1, the preprocessing is as similar to Zhao et al. (2022) as possible. However, the *JGA* is still lower than that reported by Zhao et al. (2022) (Row 1) by 1.8%. One possible difference could arise due to parser implementation: Zhao et al. (2022) do not release their parser code.

The baseline result on SGD is closer to Zhao et al. (2022) than the reproduction by Coca (2023) by 1.3%. The code used by Coca (2023) is examined and their results are confirmed independently using their setup. The difference is likely due to not treating categorical slots with only numerical candidate values as non-categorical. The performances on SGD-X are similar, so direct comparisons between `KSTRandom` and `KST-DA` will be made on SGD-X.

Model	JGA_{eval}	JGA	$JGA_{v_{1-5}}$	$JGA^{\text{seen}}_{v_{1-5}}$	$JGA^{\text{unseen}}_{v_{1-5}}$	$SS\downarrow$
D3ST (Zhao et al., 2022)	-	*72.9*	-	-	-	-
D3ST (Coca, 2023)	-	*69.8*	*56.5*	*73.6*	*50.8*	*70.1*
KST-DA (Coca, 2023)	-	*74.4*	*66.7*	*88.8*	*59.4*	*43.4*
D3ST	89.1	71.1	56.5	74.8	50.4	70.7
`KSTRandom`	87.6	**75.6**	**67.4**	88.9	**60.2**	43.5
`KSTRandomConcat`	89.7	75.5	67.2	**89.2**	59.8	**40.2**

Table 1: Results of D3ST baseline and KST augmentation on SGD dataset, and results of prior work. "-" indicates no number is available from literature. Column maxima are in **bold**

When the model is trained with the `KSTRandom` prompt format (Row 5), the test JGA on the SGD dataset is 1.2% higher than `KST-DA` (Coca, 2023, Row 3). On SGD-X, the JGA averaged across variants are slightly better, with comparable SS. Therefore, by employing random sampling of KSTs, comparable performance is achieved using a training dataset with a size 1/6 of that formed using data augmentation with manually mined KSTs, while having similar prompt lengths. This may require less computation and training time, but we could not experimentally verify this due to limited resources.

Furthermore, the JGA of `KSTRandom` on the SGD dataset is 4.5% higher than our baseline, and 2.7% higher than Zhao et al. (2022). It is within 0.7% from SDT-seq (Gupta et al., 2022) with the same number of parameters (Table 2, Row 3), whose slots contain complete dialogue and ground truth examples (as shown in Figure 9) to predict all slots in a single pass like D3ST. `KSTRandom` prompts are of similar lengths to the original D3ST prompts and can be shorter than those of SDT-seq, so the training is faster with the same dataset size. Additionally, no manual effort is required to train with `KSTRandom`, while SDT models require hand-crafted coherent yet succinct dialogues that cover all slots

(Gupta et al., 2022). While MT-SGDST (Kapelonis, Georgiou, and Potamianos, 2022) performs better on SGD, it is not robust to schema variations.

`KSTRandomConcat` (Row 6) achieves similar JGAs, while further reducing SS by 3.3. This means it is more robust to schema variations, likely due to the greater prompt diversity from a larger training dataset. Its SS is comparable to T5DST (Lin et al., 2021a), shown in Row 2 of Table 2, while having higher JGA on both SGD and SGD-X.

Model	JGA	$JGA_{v_{1-5}}$	$JGA^{\text{seen}}_{v_{1-5}}$	$JGA^{\text{unseen}}_{v_{1-5}}$	$SS\downarrow$
SGP-DST (Lee et al., 2022)	*60.5*	*49.9*	*60.7*	*46.3*	*51.9*
T5DST (Lee et al., 2022)	*72.6*	*64.0*	*79.3*	*58.9*	*40.4*
SDT-seq (Gupta et al., 2022)	*76.3*	-	-	-	-
SDT-ind (Gupta et al., 2022)	*78.2*	-	-	-	-
MT-SGDST (Coca, 2023)	*79.9*	*60.8*	*72.5*	*56.9*	*69.5*

Table 2: Performances of state-of-the-art DST models on the SGD and SGD-X datasets. The references are of the value sources if the original paper of the model does not evaluate on SGD or SGD-X

SGD unseen service transfer. As mentioned in Section 2.1.1, the SGD test set contains services unseen in the training set. To investigate the increase in SGD performances, the test JGAs are calculated separately for seen and unseen services and are shown in Table 3.

Model	JGA	JGA^{seen}	JGA^{unseen}
D3ST (Coca, 2023)	*69.8*	*92.8*	*62.2*
`KST-DA` (Coca, 2023)	*74.4*	*92.8*	*68.3*
D3ST	71.1	93.3	63.7
`KSTRandom`	75.6	93.2	69.7
`KSTRandomConcat`	75.5	93.2	69.6

Table 3: Breakdown of JGAs on the SGD dataset into seen and unseen services

`KSTRandom` and `KSTRandomConcat` have similar JGA breakdowns, both have a JGA^{unseen} around 6% higher than that of our baseline (Row 3). This demonstrates KST augmentation improves the zero-shot transfer ability of the model to unseen service schemata. There is a negligible regression in seen services, thus the improvements to overall KSTs can be attributed to the unseen services. To further analyse this behaviour, the increases in JGAs from our baseline to the `KSTRandom` models are found on a service level (Table 4).

Out of the 21 services in the SGD test set, 15 are not present in the training set. However, only 4 of these unseen services belong to unseen domains (shown in **bold** in Table 4), corresponding to 12.7% of the testing examples. Therefore, the services can be split into three categories – seen services, unseen services from seen domains, and unseen services from unseen domains. The increases in JGAs are multiplied by the number of testing examples of that service to obtain the absolute increase in joint-correctly predicted examples. The proportions of the contribution of these three categories to the total number of increased correct predictions are −0.6%, 99.8% and 0.8% respectively.

The improvements mainly come from services from seen domains because of the similarities between their schema and those of seen services from the same domain. *E.g.* the unseen `Buses_3` in the test set contains the slot `from_city` with the description "`the city to depart from`". While `Buses_1` in the train set has a slot called `from_location` with the description "`city where the bus is leaving from`". The two descriptions are paraphrases. Furthermore, it is very likely that the dialogues about these services also follow similar linguistic patterns and structures, due to the usage of the probabilistic automaton when constructing SGD and the nature of human dialogues (Shi, Zhao, and Yu, 2019). Therefore, our KST-augmented models perform well on these services. This is not the case for unseen services from unseen domains in general.

Service	Increase	Service	Increase	Service	Increase
Alarm_1	**0.13**	Media_3	16.15	Restaurants_2	1.81
Buses_3	3.67	**Messaging_1**	**−8.59**	*RideSharing_2*	*1.43*
Events_3	19.17	*Movies_1*	*−0.62*	*Services_1*	*−0.80*
Flights_4	31.51	Movies_3	1.89	Services_4	−0.40
Homes_2	−8.56	Music_3	3.9	**Trains_1**	**4.76**
Hotels_2	*−0.45*	**Payment_1**	**0.64**	*Travel_1*	*0.30*
Hotels_4	0.39	RentalCars_3	5.14	*Weather_1*	*0.49*

Table 4: Increases of JGAs of `KSTRandom` from our D3ST baseline on SGD. Seen services are *italic*, unseen services of new domains are **bold**

`KSTRandom` and `KSTRandom` have higher JGA^{seen} than `KST-DA`, indicating it is likely that treating numerical categorical slots differently contributes to the better performances, *i.e.* the gains are not purely from random sampling.

5.1.2 MultiWOZ Baseline and Data Augmentation

JGA of the reproduced D3ST baseline on the MultiWOZ dataset (72.1%) matches that from Zhao et al. (2022), as shown in Table 5. The JGA on the test set is very similar to that on the validation dataset (JGA_{eval}), which is different from SGD, where an 18% drop is observed in Table 1. This is due to the MultiWOZ test set not containing unseen domains.

Model	JGA_{eval}	JGA
D3ST (Zhao et al., 2022)	-	*72.1*
D3ST	72.7	72.1
`KSTRandom`	70.2	69.1
`KSTRandomConcat`	72.0	70.4

Table 5: Results of D3ST baseline and KST augmentation on MultiWOZ dataset. "-" indicates no number is available from literature

`KSTRandom` and `KSTRandomConcat` result in lower JGAs than the baseline. The regression is much more severe than in SGD JGA^{seen}, which is used here since the MultiWOZ test set does not have any unseen domains. This decrease may be due to the noisy annotations in MultiWOZ. Some utterances annotated with knowledge-seeking actions in MultiWOZ 2.2 are not suitable to use as alternative slot descriptions, *e.g.* the dialogue act annotation for the utterance "`actually, I'd like a guest house`" is `REQUEST(hotel-name)`.

To investigate whether the data augmentation approaches improve model zero-shot abilities and robustness to schema variation, further cross-domain transfer and unseen description transfer experiments are carried out.

MultiWOZ cross-domain transfer. Zhao et al. (2022)[4] and Lin et al. (2021b) use T5 Large (770M parameters), and are trained and evaluated on MultiWOZ 2.1. While Lin et al. (2021a) uses T5 small (60M parameters) and MultiWOZ 2.0. This means the results are not directly comparable, hence we only compare our baseline and `KSTRandomConcat` results shown in Table 6.

`KSTRandomConcat` is used because it achieves closer JGA to the baseline model on the full test set in the previous experiment. It does not produce consistent improvements or regressions across the five domains, with higher JGA^{unseen} only in Attraction and Taxi domains. The JGAs of models trained with different random seeds exhibit large variances – the standard deviation between results from models using the original D3ST prompts with the Taxi domain left out is 11%, while that between the three models with `KSTRandomConcat` prompts and Restaurant domain left out is as large as 12%. Therefore, no conclusion with statistical significance can be reliably made.

Model	Attraction	Hotel	Restaurant	Taxi	Train
D3ST	79.2	**28.9**	**43.9**	58.9	**42.1**
`KSTRandomConcat`	**81.4**	24.5	32.8	**73.4**	36.9

Table 6: JGA^{unseen} when leaving the domain out during training. Column maxima are in **bold**

Upon inspection, the most common type of error is the model omitting slots, which is also reported by Campagna et al. (2020). These partially correct predictions are not rewarded when evaluating with JGA.

Even though there are shared slots between domains in MultiWOZ, their descriptions are not paraphrases, *e.g.* the descriptions of slot `area` are "`area or place of the hotel`" in the Hotel domain and "`area or place of the attraction`" in the Attraction domain. This is in a way similar to the unseen services from unseen domains in SGD mentioned earlier. The slot descriptions and dialogues from different domains may be too diverse and inconsistent for KST augmentation to be effective, as the augmentation only increases robustness to limited description variations.

MultiWOZ unseen description transfer. The baseline and KST augmented models analysed earlier are decoded using prompts formed from the alternative slot descriptions (as mentioned in Section 4.2.3), the results are shown in Table 7. The `KSTRandom` and `KSTRandomConcat` models perform significantly better than the baseline, with reductions in JGAs of 2.3% and 4.9% when compared with the results in Table 5. On the other hand,

[4]The model size used is not reported in the paper and is confirmed through communication with the author.

that of the baseline model is decreased by 46.8%. This demonstrates the increased schema robustness of the augmented models.

Model	*JGA*
D3ST	25.3
`KSTRandom`	66.8
`KSTRandomConcat`	65.5

Table 7: Results of MultiWOZ baseline and KST augmented models decoded using alternative slot descriptions

The models trained under the leave-one-domain-out setup are also decoded with the alternative slot descriptions (Table 8). The `KSTRandomConcat` model performs better than the baseline in four out of the five domains with smaller variances.

Model	Attraction	Hotel	Restaurant	Taxi	Train
D3ST	44.8	17.0	**26.5**	2.6	15.4
`KSTRandomConcat`	**77.2**	**18.0**	19.0	**4.3**	**25.5**

Table 8: JGA^{unseen} when the leave-one-domain-out models are decoded with alternative slot descriptions. Column maxima are in **bold**

Limitations of MultiWOZ. Due to the lack of active domain annotations, descriptions from all slots in the ontology are included in the D3ST prompts. This is not scalable, as new domains will further increase the encoder input length, leading to excessive truncation.

5.2 Prompt Grounding and Random KST Sampling

5.2.1 Prompt Grounding

The results when the prompts are grounded by KSTs or both slot/intent names and KSTs during training and decoding are shown below in Table 9. `RandomTurnSlot` performs the best on SGD-X, with the lowest SS at 22.7. This is further reduced from 23.7 of `TurnSlot` (Coca, 2023). It also has better performance than `TurnSlot` on all other metrics, demonstrating the effectiveness of random KST sampling.

Model	JGA	JGA^{seen}	JGA^{unseen}	$JGA_{v_{1-5}}$	$JGA^{\text{seen}}_{v_{1-5}}$	$JGA^{\text{unseen}}_{v_{1-5}}$	$SS \downarrow$
`Turn`[†]	*75.8*	*92.9*	*70.1*	*69.5*	*88.5*	*63.2*	*36.6*
`TurnSlot`[†]	*74.7*	*92.8*	*68.7*	*72.0*	*90.7*	*65.6*	*23.7*
`RandomTurn`	**77.0**	**93.2**	**71.6**	67.6	83.1	62.5	39.7
`RandomTurnSlot`	75.1	93.1	69.2	**72.1**	**90.8**	**65.9**	**22.7**

Table 9: Results of models using `RandomTurn` and `RandomTurnSlot` both in training and decoding on SGD dataset, and results of Coca (2023) (annotated with †). Column maxima are in **bold**

RandomTurn achieves the best performance on the SGD dataset, but also the worst on SGD-X. Especially, the average seen JGA on SGD-X is 5.4% lower than that of Turn (Coca, 2023). This is unexpected, as all models trained using randomly sampled KSTs on SGD so far demonstrate greater robustness than the corresponding ones trained with five manually sampled turns. The RandomTurn models show greater degradation of performance on v4 and v5 variants of SGD-X when compared to other variants, than the RandomTurnSlot models. Upon further investigation on v5, 19% of all turns are predicted incorrectly because the model omits slots or predicts slots not in the ground truth. This is higher than RandomTurnSlot on v5, at 5%. The reason is unclear, so decoding is repeated, giving the same results. The possibility of errors during fine-tuning cannot be completely ruled out and should be investigated further in future work.

5.2.2 Grounded Decoding

When comparing the results in Table 9 with those of models trained with KST augmentation in Table 1, the models with grounded prompts during both training and decoding achieve comparable results on SGD and better results on SGD-X. The KST augmented models are decoded with the same grounded prompts, and the results are tabulated in Table 10.

The ground decoding approach results in improvements in SGD-X performances of KSTRandom and KSTRandomConcat for all combinations between training data augmentation approaches and decoding prompt formats, even though the training and decoding prompt formats do not match. This is because the KSTs in the decoding prompts "facilitates knowledge sharing between seen and unseen slots" (Coca, 2023). KSTRandom has a lower SS when decoded with Turn prompts (Row 3), while KSTRandomConcat has a lower SS with TurnSlot prompts (Row 6).

	Model	*JGA*	$JGA_{v_{1-5}}$	$JGA^{\text{seen}}_{v_{1-5}}$	$JGA^{\text{unseen}}_{v_{1-5}}$	$SS \downarrow$
1	*KST-DA/Turn (Coca, 2023)*	*74.9*	*71.7*	***91.9***	*65.0*	*30.7*
2	*KST-DA/TurnSlot (Coca, 2023)*	*73.8*	*71.0*	*91.0*	*64.3*	*31.2*
3	KSTRandom/Turn	**75.9**	**73.5**	91.8	**67.4**	**26.1**
4	KSTRandom/TurnSlot	74.3	72.7	91.7	66.4	26.7
5	KSTRandomConcat/Turn	75.7	71.7	90.6	65.4	30.3
6	KSTRandomConcat/TurnSlot	74.5	71.6	90.2	65.4	28.4
7	KSTRandom/D3ST	75.6	67.4	88.9	60.2	43.5
8	KSTRandomConcat/D3ST	75.5	67.2	89.2	59.8	40.2

Table 10: Results from decoding various models with grounded prompts. "/" separates training and decoding prompt formats. The bottom two rows are copied from Table 1 to facilitate comparison. Column maxima are in **bold**

On the other hand, decoding with Turn prompts improves SGD performances for both models compared to decoding with the original D3ST prompts (Rows 7 and 8), while TurnSlot decoding causes regressions on SGD. The reduction is mainly attributed to lower JGAs on unseen services, as shown in Table 11. This is likely due to the slot names in SGD schemas containing ambiguous information (Coca, 2023; Zhao et al., 2022). All grounded decoding gives slightly worse seen JGA, likely due to the prompt format mismatch.

Decoding KSTRandom with Turn prompts achieves the best results, with a 6.1% increase in the average JGA on SGD-X and a 17.4 decrease in SS than not grounding the decoding prompts. It also gives higher JGAs on both SGD and SGD-X than `RandomTurnSlot` (Table 9). However, it has a higher SS, meaning the models predict more slots correctly but are less consistent across the schema variants.

Furthermore, both grounded decoding methods achieve higher or comparable results in all metrics on models trained with randomly sampled KSTs than the ones with manual KST mining (Coca, 2023). The worst-performing combination is still comparable to the best `KST-DA` decoding results.

Model	*JGA*	*JGA*$^{\text{seen}}$	*JGA*$^{\text{unseen}}$
KST-DA/Turn (Coca, 2023)	*74.9*	*92.6*	*69.0*
KST-DA/TurnSlot (Coca, 2023)	*73.8*	*92.5*	*67.6*
`KSTRandom/Turn`	75.9	93.0	70.2
`KSTRandom/TurnSlot`	74.3	93.1	68.0
`KSTRandomConcat/Turn`	75.7	93.0	69.9
`KSTRandomConcat/TurnSlot`	74.5	92.7	68.4
`KSTRandom/D3ST`	75.6	93.2	69.7
`KSTRandomConcat/D3ST`	75.5	93.2	69.6

Table 11: Breakdown of JGAs when decoding with grounded prompts on the SGD dataset into seen and unseen services. The bottom two rows are taken from Table 3 to facilitate comparison

5.3 Cross-Dataset Transfer

The results of applying models trained on one of the SGD and MultiWOZ datasets and decoded on the other are shown in Table 12. The results reported by Zhao et al. (2022) are based on T5 XXL models (11B parameters), at 28.9% from SGD to MultiWOZ and 23.1% vice versa. Our SGD to MultiWOZ baseline results obtained using a much smaller model are closer to the corresponding value from Zhao et al. (2022) than the results from MultiWOZ to SGD. The low JGAs indicate diverse structures of the schemata and dialogues, but the asymmetry is interesting. We hypothesise that it is due to the MultiWOZ dataset only containing 30 slots across five domains, making generalisation difficult.

Models trained with `KSTRandomConcat` on SGD show a 0.4% increase in JGA compared to the baseline when evaluated on MultiWOZ. However, regression is observed on SGD JGA with `KSTRandomConcat` models trained on MultiWOZ.

Transfer	*JGA*
SGD D3ST $\rightarrow$ MultiWOZ	18.3
SGD `KSTRandomConcat` $\rightarrow$ MultiWOZ	18.7
MultiWOZ D3ST $\rightarrow$ SGD	6.1
MultiWOZ `KSTRandomConcat` $\rightarrow$ SGD	4.2

Table 12: Results of cross-dataset transfer

6 Conclusion

Replacing the D3ST training prompts with randomly sampled KSTs (`KSTRandom`) achieves SGD performance competitive with state-of-the-art DST models. It has comparable performances on both SGD and SGD-X to model augmented with manually mined KSTs (Coca, 2023) while using a much smaller training set. Using this dataset to augment the training set with D3ST prompts (`KSTRandomConcat`) further improves the robustness to schema variations, beyond what is achievable with manually mined KST augmentation.

Improvements to the SGD JGA are attributed to the better performance on unseen slots from seen domains, whose schema descriptions are paraphrases of those of seen services from the same domain. The increased schema robustness allows the model to exploit this uniformity, resulting in the higher zero-shot unseen service transfer performance.

No improvement is observed when random KST augmentation is applied to the MultiWOZ dataset, as its test set does not contain any unseen domains. Instead, a regression is observed when compared to our MultiWOZ baseline, likely due to noisy annotations of the dataset. Furthermore, no improvement to MultiWOZ cross-domain evaluation results is observed, due to the diverse and inconsistent schemata and dialogues between domains. However, decoding the augmented model using alternative slot descriptions on MultiWOZ shows improved schema robustness, validating the applicability of KST augmentation to the MultiWOZ dataset.

Decoding with prompts grounded by KSTs gives the best results, as they allow knowledge sharing between seen and unseen slots (Coca, 2023). All grounded decoding experiments show better schema robustness using models trained with random KST sampling and the ones with manual KST mining. Except for the models trained and decoded using turn-grounded prompts, which show a regression with unclear reasons.

The better schema robustness compared to Coca (2023) demonstrates the effectiveness of random KST sampling to further increase prompt diversity than manual KST mining.

KST augmentation improves SGD to MultiWOZ cross-dataset transfer ability slightly, but not vice versa. Models trained on SGD have higher JGAs when evaluating on MultiWOZ than the other way around, likely due to the limited number of slots in the MultiWOZ dataset.

6.1 Limitations and Future Work

The reasons for the regression when training and decoding D3ST with the `RandomTurn` prompt format are unclear and should be the subject of future investigations.

The sampled KSTs often contain irrelevant parts before the actual questions due to the nature of human dialogues. A heuristic is used to extract the substring containing the questions. However, it is not guaranteed to extract the best substring under all circumstances. The linguistic diversity of sampled KSTs is not explicitly analysed.

Moreover, methods of automatically generating the KST-grounded prompts for decoding of unseen services in the real world are not discussed. In future work, large language models can be used to generate their KSTs automatically (Coca, 2023).

References

Apache/Arrow (2023). *Apache/Arrow: Apache Arrow Is a Multi-Language Toolbox for Accelerated Data Interchange and in-Memory Processing.* URL: https://github.com/apache/arrow (visited on 05/18/2023).

Balaraman, Vevake, Seyedmostafa Sheikhalishahi, and Bernardo Magnini (2021). "Recent Neural Methods on Dialogue State Tracking for Task-Oriented Dialogue Systems: A Survey". In: *Proceedings of the 22nd Annual Meeting of the Special Interest Group on Discourse and Dialogue.* SIGDIAL 2021. Singapore and Online: Association for Computational Linguistics, pp. 239–251. URL: https://aclanthology.org/2021.sigdial-1.25 (visited on 10/27/2022).

Budzianowski, Paweł et al. (2018). "MultiWOZ – A Large-Scale Multi-Domain Wizard-of-Oz Dataset for Task-Oriented Dialogue Modelling". arXiv: 1810.00278. URL: http://arxiv.org/abs/1810.00278 (visited on 10/13/2022).

Campagna, Giovanni et al. (2020). "Zero-Shot Transfer Learning with Synthesized Data for Multi-Domain Dialogue State Tracking". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics.* ACL 2020. Online: Association for Computational Linguistics, pp. 122–132. DOI: 10.18653/v1/2020.acl-main.12. URL: https://aclanthology.org/2020.acl-main.12 (visited on 10/22/2022).

Chen, Hongshen et al. (2017). "A Survey on Dialogue Systems: Recent Advances and New Frontiers". In: *ACM SIGKDD Explorations Newsletter* 19.2, pp. 25–35. ISSN: 1931-0145. DOI: 10.1145/3166054.3166058. URL: https://doi.org/10.1145/3166054.3166058 (visited on 10/28/2022).

Coca, Alexandru (2023). *Grounding Description-Driven Dialogue State Trackers with Knowledge-Seeking Turns.* preprint.

Cormen, Thomas H. et al. (2001). *Introduction to Algorithms.* 2nd ed. The MIT Press. ISBN: 0-262-03293-7.

Eric, Mihail et al. (2020). "MultiWOZ 2.1: A Consolidated Multi-Domain Dialogue Dataset with State Corrections and State Tracking Baselines". In: *Proceedings of the Twelfth Language Resources and Evaluation Conference.* LREC 2020. Marseille, France: European Language Resources Association, pp. 422–428. ISBN: 979-10-95546-34-4. URL: https://aclanthology.org/2020.lrec-1.53 (visited on 05/28/2023).

Gao, Jianfeng, Michel Galley, and Lihong Li (2019). *Neural Approaches to Conversational AI.* DOI: 10.48550/arXiv.1809.08267. arXiv: 1809.08267 [cs]. URL: http://arxiv.org/abs/1809.08267 (visited on 05/31/2023). preprint.

Gupta, Raghav et al. (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.* NAACL-HLT 2022. Seattle, United States: Association for Computational Linguistics, pp. 4541–4549. DOI: 10.18653/v1/2022.naacl-main.336. URL: https://aclanthology.org/2022.naacl-main.336 (visited on 10/28/2022).

Han, Ting et al. (2020). *MultiWOZ 2.3: A Multi-Domain Task-Oriented Dialogue Dataset Enhanced with Annotation Corrections and Co-Reference Annotation.* arXiv.org. URL: https://arxiv.org/abs/2010.05594v3 (visited on 05/25/2023).

Jacqmin, Léo, Lina M. Rojas Barahona, and Benoit Favre (2022). ""Do You Follow Me?": A Survey of Recent Approaches in Dialogue State Tracking". In: *Proceedings of the 23rd Annual Meeting of the Special Interest Group on Discourse and Dialogue.* SIGDIAL

2022. Edinburgh, UK: Association for Computational Linguistics, pp. 336–350. URL: https://aclanthology.org/2022.sigdial-1.33 (visited on 05/28/2023).

Kapelonis, Eleftherios, Efthymios Georgiou, and Alexandros Potamianos (2022). *A Multi-Task BERT Model for Schema-Guided Dialogue State Tracking*. DOI: 10.48550/arXiv.2207.00828. arXiv: 2207.00828 [cs]. URL: http://arxiv.org/abs/2207.00828 (visited on 05/29/2023). preprint.

Kelley, J. F. (1984). "An Iterative Design Methodology for User-Friendly Natural Language Office Information Applications". In: *ACM Transactions on Information Systems* 2.1, pp. 26–41. ISSN: 1046-8188. DOI: 10.1145/357417.357420. URL: https://dl.acm.org/doi/10.1145/357417.357420 (visited on 05/28/2023).

Lee, Chia-Hsuan, Hao Cheng, and Mari Ostendorf (2021). *Dialogue State Tracking with a Language Model Using Schema-Driven Prompting*. DOI: 10.48550/arXiv.2109.07506. arXiv: 2109.07506 [cs]. URL: http://arxiv.org/abs/2109.07506 (visited on 10/28/2022). preprint.

Lee, Harrison et al. (2022). "SGD-X: A Benchmark for Robust Generalization in Schema-Guided Dialogue Systems". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.10 (10), pp. 10938–10946. ISSN: 2374-3468. DOI: 10.1609/aaai.v36i10.21341. URL: https://ojs.aaai.org/index.php/AAAI/article/view/21341 (visited on 10/28/2022).

Lin, Zhaojiang et al. (2021a). "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*. NAACL-HLT 2021. Online: Association for Computational Linguistics, pp. 5640–5648. DOI: 10.18653/v1/2021.naacl-main.448. URL: https://aclanthology.org/2021.naacl-main.448 (visited on 10/28/2022).

Lin, Zhaojiang et al. (2021b). *Zero-Shot Dialogue State Tracking via Cross-Task Transfer*. DOI: 10.48550/arXiv.2109.04655. arXiv: 2109.04655 [cs]. URL: http://arxiv.org/abs/2109.04655 (visited on 03/02/2023). preprint.

Nekvinda, Tomáš and Ondřej Dušek (2021). "Shades of BLEU, Flavours of Success: The Case of MultiWOZ". In: *GEM 2021 - 1st Workshop on Natural Language Generation, Evaluation, and Metrics, Proceedings*, pp. 34–46. DOI: 10.48550/arxiv.2106.05555. arXiv: 2106.05555. URL: https://arxiv.org/abs/2106.05555v1 (visited on 10/13/2022).

Raffel, Colin et al. (2020). "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *Journal of Machine Learning Research* 21.140, pp. 1–67. ISSN: 1533-7928. URL: http://jmlr.org/papers/v21/20-074.html (visited on 10/28/2022).

Rastogi, Abhinav et al. (2019). "Towards Scalable Multi-domain Conversational Agents: The Schema-Guided Dialogue Dataset". arXiv: 1909.05855. URL: http://arxiv.org/abs/1909.05855 (visited on 10/13/2022).

Shah, Pararth et al. (2018). *Building a Conversational Agent Overnight with Dialogue Self-Play*. arXiv: 1801.04871 [cs]. URL: http://arxiv.org/abs/1801.04871 (visited on 05/26/2023). preprint.

Shi, Weiyan, Tiancheng Zhao, and Zhou Yu (2019). *Unsupervised Dialog Structure Learning*. DOI: 10.48550/arXiv.1904.03736. arXiv: 1904.03736 [cs]. URL: http://arxiv.org/abs/1904.03736 (visited on 05/31/2023). preprint.

Wolf, Thomas et al. (2019). “HuggingFace’s Transformers: State-of-the-art Natural Language Processing”. arXiv: 1910.03771. URL: http://arxiv.org/abs/1910.03771 (visited on 10/13/2022).

Wu, Chien-Sheng et al. (2019). *Transferable Multi-Domain State Generator for Task-Oriented Dialogue Systems*. DOI: 10.48550/arXiv.1905.08743. arXiv: 1905.08743 [cs]. URL: http://arxiv.org/abs/1905.08743 (visited on 10/28/2022). preprint.

Ye, Fanghua, Jarana Manotumruksa, and Emine Yilmaz (2022). *MultiWOZ 2.4: A Multi-Domain Task-Oriented Dialogue Dataset with Essential Annotation Corrections to Improve State Tracking Evaluation*. DOI: 10.48550/arXiv.2104.00773. arXiv: 2104.00773 [cs]. URL: http://arxiv.org/abs/2104.00773 (visited on 01/23/2023). preprint.

Zang, Xiaoxue et al. (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*. NLP4ConvAI 2020. Online: Association for Computational Linguistics, pp. 109–117. DOI: 10.18653/v1/2020.nlp4convai-1.13. URL: https://aclanthology.org/2020.nlp4convai-1.13 (visited on 02/08/2023).

Zhao, Jeffrey et al. (2022). *Description-Driven Task-Oriented Dialog Modeling*. DOI: 10.48550/arXiv.2201.08904. arXiv: 2201.08904 [cs]. URL: http://arxiv.org/abs/2201.08904 (visited on 10/28/2022). preprint.

Appendix A Implementation Details

The project started with two code bases inherited from the research group, one implementing D3ST on the SGD dataset used by Coca (2023), and another one written to reproduce T5DST (Lee, Cheng, and Ostendorf, 2021) by Coca and Tseng. The new implementation is based on the T5DST code infrastructure, extended extensively to support D3ST on both SGD and MultiWOZ datasets and random KST sampling, while keeping backward compatibility with the T5DST model.

In this section, the design and implementation of the systems will be discussed in detail, focusing on the code developed since the beginning of this project.

A.1 Data Preprocessing

SGD and MultiWOZ datasets are first preprocessed into a dataset-agnostic data format used by the Robust-DST system. The preprocessing scripts are modified from the newly released code[5] from Zhao et al. (2022).

A.1.1 Dataset-Agnostic Data Format

A new data format is designed to serve as a dataset-agnostic input to the Robust-DST system, supporting both SGD and MultiWOZ datasets and allowing random KST sampling.

Each training/evaluation example is stored as a row according to the JSON Lines format, which allows the usage of the *HuggingFace* (Wolf et al., 2019) `datasets` library for efficient data loading and processing.

Example. An example of preprocessed data for one frame of a turn of a dialogue is shown in Listing 1. The mappings are stored as serialised strings to allow loading the dataset as an *Arrow* table (*Apache/Arrow* 2023, used by the `datasets` library), which otherwise will encode the mappings differently by padding each row so they contain the same keys. `description_mapping` is a mapping from slot index to slot description. Storing the prompt in this way allows easy replacement of schema descriptions to sampled KSTs (more details in Section A.3). `slot_mapping`, `cat_values_mapping`, `intent_mapping` and other metadata are used by the parser (Section A.2) to parse model outputs into a format accepted by the scoring scripts. `prev_sys_utt`, `prev_sys_acts` *etc.* are used to sample the appropriate KST (Section A.3), both of which are present because state annotations are only present after each complete user-system interaction.

```
{
  "description_mapping": "{\"0\": \"how many stops the route has 0a) direct 0b) one-stop\"}",
  "state": "[states] 0=0a [intents] i0 [req_slots] ",
  "dialogue_context": "[user] can you assist me in finding a bus, i would like a direct one.",
  "slot_mapping": "{\"0\": \"category\"}",
  "cat_values_mapping": "{\"category\": {\"direct\": \"0a\", \"one-stop\": \"0b\"}}",
  "intent_mapping": "{\"i0\": \"FindBus\", \"i1\": \"BuyBusTicket\"}",
  "turn_domain": "Buses_3",
  "turn_idx": "0",
  "dialogue_id": "4_00101",
  "file_name": "dialogues_004.json",
  "sys_utt": "",
  "sys_acts": "{}",
  "user_utt": "can you assist me in finding a bus, i would like a direct one.",
  "user_acts": "{\"INFORM\": [\"category(direct)\"], \"INFORM_INTENT\": [\"FindBus\"]}"
```

[5]https://github.com/google-research/task-oriented-dialogue/tree/main/state_tracking/d3st

```
},
```

Listing 1: A example row of the preprocessed data

Difference to Coca, 2023. This data format is very different from that of the D3ST preprocessing code from Coca (2023), which follows the structure of the raw SGD datasets – a nested dictionary. The new code flattens this nested structure, allowing the sampling of turns and extensibility to support the MultiWOZ dataset. The dialogue context is built during preprocessing, instead of building it during training, greatly simplifying data loading code. More information is also included to allow KST sampling.

A.1.2 SGD Preprocessing

Zhao et al. (2022) use a Python `dataclass` called `TurnInfo` to keep state across turns and frames in a dialogue, and outputs strings of preprocessed training inputs (prompts and dialogue context) and targets. However, the preprocessed data do not contain the information required to parse and score the model outputs or to enable sampling of KSTs, so more states are added to `TurnInfo` and tracked (*e.g.* mappings, turn domain, utterance acts).

The output logic is also modified. A new `dataclass` is added, it is initialised with a `TurnInfo` object with serialisation applied. This new `dataclass` is converted into a Python dictionary, which forms one row in the output data (*i.e.* the columns in the preprocessed dataset correspond to the attributes of this class). This allows the generation of data in the desired format while keeping the change to the code from Zhao et al. (2022) to a minimum, which ensures accurate reproduction of the published results.

However, some modifications are made.

Accumulate slots. Interestingly Zhao et al. (2022) accumulate the slots throughout a dialogue, instead of resetting the states to the ground truth of the dataset. This means no slots are removed from the target state, even if they are removed in the underlying ground truth dataset.

Options are added to toggle these behaviours for investigation of their effects. Training is done using the Robust-DST codebase with that option enabled during preprocessing, while keeping training parameters the same as Zhao et al. (2022). The results are shown in Table 13.

The results with and without accumulating slots are very similar, so analysis of the occurrence of a slot being removed as dialogue progress in the raw dataset is performed. It only happens in the `Banks_1` service in the training dataset and `Payments_1` in the test dataset. All of these dialogues involve switching back to intent in earlier turns. For example, in dialogue `32_00011` of the SGD training dataset, the active intent changes from `CheckBalance` to `TransferMoney`, and back to `CheckBalance` (on which turn the states are reset).

In the training dataset, this only happens for 472 out of the 175780 examples, so accumulating the slots should not have a large effect on the overall JGA. However, the performance of the `Payments_1` service may be negatively affected. As not accumulating the slots produces preprocessed dataset following the raw dataset (the ground truth) exactly, it is chosen as the preprocessing method. It is unclear whether accumulating slots is done intentionally by Zhao et al. (2022).

	JGA_{orig}	JGA_{v1-5}	JGA^{seen}_{v1-5}	$JGA^{\text{unseen}}_{v1-5}$
- (Zhao et al., 2022)	*72.9*	-	-	-
Not accumulate slots	71.1	56.5	74.8	50.4
Accumulate slots	70.1	56.9	75.8	50.6

Table 13: Baseline D3ST results on the SGD dataset, *italic row* is taken from literature. Three-run averages

Other differences in preprocessing. Other than not accumulating slots, the following changes are made because the parser from Coca (2023) is used,

1. The value separator, which is used to separate possible values in the prompt, used is different from the default `" | "`. This is due to the dataset containing the slot-value pair `"restaurant_name": ["Ginza | Japanese Sushi Restaurant"]`, which causes parsing difficulties.

2. Instead of `":"`, `"="` is used as the delimiter between slot/intent indices and slot/intent descriptions/values in prompts and targets. This is to avoid ambiguity with the time separator in slot values.

An error in the SGD preprocessing code from (Zhao et al., 2022) is fixed and is discussed in Section B.1.

A.1.3 MultiWOZ Preprocessing

A modified preprocessing script based on that from Zhao et al. (2022) is used to process the MultiWOZ 2.4 dataset. However, files from other versions of MultiWOZ are also used for the schema, slot descriptions and dialogue acts (as discussed in Section 4.2).

The preprocessing script is extended to process dialogue acts. If the action parameters contain slot values, a simple heuristic is used to align them with the schema. Similar to preprocessing of the SGD dataset (Section A.1.2), the value separator and index delimiter are modified, and other necessary metadata are added. The error mentioned in Section B.2 is fixed.

A.2 Parser

The parsing is done using regular expressions, together with `slot_mapping`, `intent_mapping` and `cat_values_mapping` from the preprocessed data to map the predicted index back to the slot name, categorical values and intent name.

The parser from the old D3ST code base is used and integrated with the T5DST parser. A new `Parser` class is designed, which serves as the parent class of T5DST and D3ST parsers, employing the Factory Method design pattern. This provides a common interface and allows for code reuse and configuration encapsulation.

The integrated parser is tested by parsing the ground truth states and checking whether the resulting JGA is 1.

A.3 Runtime Preprocessor and Random KST Sampling

A.3.1 Processing Pipeline

Similar to the parsers (Section A.2), the T5DST and D3ST preprocessors are both subclasses of a `Preprocessor` base class, which provides helper methods for tokenisation and more.

To enable a flexible and extensible way of defining preprocessors, a lightweight pipeline library[6] is developed and published for direct installation from the Python Package Index. It supports conditional branching for building dynamic pipelines.

The pipeline library has two main components – a wrapper class for Python `Callables` that are registered as processing steps, and a class providing methods to build and execute the pipeline. The wrapper class implements the Python data descriptor interface, which enables the correct invocation of both unbound methods (static methods) and bound methods. Furthermore, the wrapper class contains the metadata required to build the pipeline, *e.g.* next processing steps, and whether to execute the current processing step.

After the processing steps are registered, a modified version (Listing 2) of the standard topological sort (Cormen et al., 2001) is used to determine the correct execution order of the processing steps, by considering the dependencies between steps and whether a step should be executed. An exception will be raised if the dependency relationships do not form a valid Directed Acyclic Graph (DAG). The steps with no dependencies that are executed will be pruned from the DAG.

```
TopologicalSort($S$)
    $P$ = MakeStack()
    foreach node $s \in S$
        if $s$.colour $\neq$ BLACK
            $Q$ = MakeStack()
            Push($Q$, $s$)
            while $Q$.size $> 0$
                $n$ = Pop($Q$)
                if $n$.colour == WHITE:
                    $n$.colour = GREY
                    Push($Q$, $n$)
                    foreach next node $x \in n$.successors
                        if $x$.colour == GREY AND $x$.torun
                            Push($Q$, $x$)
                        else if $x$.colour == GREY
                            Cycle detected error
                else
                    if $n$.colour == GRAY AND $n$.torun:
                        $n$.colour = BLACK
                        Push($P$, $n$)
    return $P$
```

Listing 2: Pseudocode of the topological sort algorithm used by the pipeline library

Finally, as the processing steps are executed sequentially, their outputs are stored such that later steps can access them. If multiple steps that are to be executed depend on a common previous step, the previous outputs are deep-copied before being passed as inputs.

[6] https://github.com/WeixuanZ/methodflow

This is necessary to prevent errors due to unintentional side-effects, since Python passes objects by reference.

The pipeline library serves as a solid foundation for the runtime preprocessor (Figure 6), which can be simply defined as shown in Listing 3. Methods of the `D3STPreprocessor` class are registered as processing steps using the `op` decorator, with an optional predicate name. The dependencies are specified through names of the functional parameters, which are replaced with the outputs of the specified step during execution. This provides encapsulation and promotes modularity, allowing new processing steps to be easily inserted into the pipeline.

```python
class D3STPreprocessor(Preprocessor):
    # ...

    @PipelineMixin.op()
    def _do_json_loads(self, examples: Batch | dict) -> Batch | dict:
        # ...
        return examples

    @PipelineMixin.op(condition="_sample_corpus")
    def _do_sample_corpus(
        self, results_from__do_json_loads: Batch | dict
    ) -> list[dict[str, str]]:
        # ...
        return new_desc_mappings

    @PipelineMixin.op(condition="_sample_table")
    def _do_sample_table(
        self, results_from__do_json_loads: Batch | dict
    ) -> list[dict[str, str]]:
        # ...
        return new_desc_mappings

    @PipelineMixin.op()
    def _do_combine_kst(
        self,
        results_from__do_json_loads: Batch | dict,
        results_from__do_sample_corpus: list[dict],
        results_from__do_sample_table: list[dict],
    ) -> Batch | dict:
        # ...
        return results_from__do_json_loads

    @PipelineMixin.op()
    def _do_join_description_mapping(
        self,
        results_from__do_combine_kst: Batch | dict,
    ) -> Batch | dict:
        # ...
        return joined_mapping
```

Listing 3: Skeleton of Python code of the pipeline

A.3.2 Efficient Random KST Sampling

Action cache. To determine the valid rows to sample the knowledge-seeking turns for a particular slot or intent, an action cache is first built. The cache is an in-memory hashtable in the form `cache[slot_or_intent][domain][slot_intent_name] -> list[tuple[row_index, speaker]]`. This reduces the quadratic time complexity with a naive nested iteration into linear time. Since only the row indices and speakers are stored, the size of the hashtable is small enough to be stored in memory, giving superb access speed.

To build the cache, each row of the dataset is iterated through. For each row, `sys_acts` and `user_acts` (examples shown in Listing 1, lines 13 and 15) are checked for the existence and uniqueness of the three knowledge-seeking actions.

Sampling. When forming the prompt, each slot/intent index and description pair of `description_mapping` of each example (*i.e.* a row, Listing 1 is an example) in the dataset is iterated through. For each index, the corresponding slot or intent name is retrieved from `slot_mapping` and `intent_mapping` respectively. Then using the name and the current turn domain, the list of valid row indices and speakers that can be sampled from can be retrieved from the cache. Finally, a random index speaker pair can be selected and its corresponding utterance (one of `sys_utt` and `user_utt`) is returned.

The sampled utterance may not contain only the desired question, such as `yes, that's right, what is the address of the restaurant?` and `perfect! i want to book a table here on march 13th.` Another regular expression `(.*[.,?!]\s)*(` ↪ `and\s)?(.*)[?.]?` is used to extract the KSTs.

Incorporation of sampled KSTs. Preprocessed descriptions for categorical slots also contain the possible values, for example, slot `0` in Listing 1: `how many stops the route has 0a) direct 0b) one-stop`. A regular expression `.*?(\s?\d+\w\).*)` matching `0a) direct 0b) one-stop` is used to extract the possible values, which are then used when incorporating the sampled KSTs.

A.4 MultiWOZ Evaluator

The official MultiWOZ evaluator from Nekvinda and Dušek (2021) is extended[7] to support DST evaluation on the MultiWOZ 2.4 dataset, and to support calculating JGA per domain.

Another widely used approach for MultiWOZ evaluation is using the TRADE preprocessing and evaluation scripts (Wu et al., 2019). It is not used here because Zhao et al. (2022) does not apply TRADE preprocessing to MultiWOZ 2.4, and the evaluation script is more difficult to integrate with the parser. Developing a script to transform MultiWOZ dataset into the SGD format, and then using the SGD parser and evaluator to obtain the metrics was also considered. Due to the difficulties in developing an accurate and reliable transformation script, the approach was not taken.

Limitations. The original MultiWOZ evaluator only evaluates against the first value if a slot has multiple possible values. This is different from the official SGD evaluation script (Rastogi et al., 2019), which returns the highest fuzzy string match score of the references and hypothesis. This may lead to the model performance being underestimated. Especially since there are 6 (out of 7372) turns in the test set, where the first possible ground truth slot value of at least one categorical slot is not a valid candidate specified in the schema used.

All slot values are evaluated using fuzzy matching, unlike in SGD evaluation (Rastogi et al., 2019). This is again due to how MultiWOZ is developed – without schemata, but also means when MultiWOZ is used in a schema-guided fashion, using this evaluator will potentially overestimate categorical slot performances.

MultiWOZ 2.4 evaluation. Originally, the evaluator only supports MultiWOZ 2.2, which is formatted according to the SGD format. The dataset is loaded and each dialogue is transformed into a list of mappings from the domain-slot name to the ground truth slot

[7] https://github.com/WeixuanZ/MultiWOZ_Evaluation

values. For MultiWOZ 2.4, a new dataset loader is developed following the same format, so the evaluation logic is not modified.

When loading, slots annotated with "none" and "not mentioned" are removed. "none" is used when a slot is not mentioned or its value has been deleted. This is confirmed by communication with one of the authors of Ye, Manotumruksa, and Yilmaz (2022). Furthermore, slot values under the "booked" fields are also removed, since they denote booking confirmations instead of dialogue states. This is the same approach taken by Zhao et al. (2022) when preprocessing MultiWOZ for D3ST.

Attention is given to ensuring the new loader follows the same logic as that of MultiWOZ 2.2, including following the original approach where only the first value is taken if a slot has multiple values. MultiWOZ uses logical expressions to represent the dialogue state (Zang et al., 2020), *e.g.* "spanish|portuguese" for no preference and "monday<thursday" for preferring thursday over monday. These strings are split on the delimiter and the first substring is always used, ignoring the logical structure, since these examples are disproportionally rare for the model to correctly learn the complex annotation. D3ST (Zhao et al., 2022) also ignores this logical structure.

On the other hand, MultiWOZ 2.2 appends the string "book" to MultiWOZ 2.1 booking slots, *e.g.* "hotelbookday". This behaviour is not replicated when loading MultiWOZ 2.4.

Leave-one-domain-out evaluation. Leave-one-domain-out evaluation is supported by filtering turns from both the hypotheses to score and the ground truth reference before metric calculations. As mentioned in Section 3.3.3, turns with empty belief states are also considered when calculating JGA under two situations:

- The turn is at the start of a dialogue related to the unseen domain
- The turn belongs to a dialogue with domain switching, and the switching to the unseen domain occurs after the given turn

The selection of turns to evaluation is achieved using the algorithm shown in Listing 4. The turns with empty ground truth belief states are selected tentatively, and removed later if the dialogue is determined to be unrelated to the unseen domain.

```
SelectTurns(hypotheses $H$, references $R$, unseen domain $u$)
    foreach dialogue $d$ in $H$
        foreach turn hypothesis $x$ in $H[d]$, corresponding reference $y$ in $R[d]$
            Remove all slots not from $u$ from $x$, $y$
            if $y$.size == 0 AND $y$.size > 0 before removing slots
                if all previous turn reference $z$ in $R[d]$ has $z$.size == 0
                    Remove all previous turns from $H[d]$, $R[d]$
                Remove $x$ from $H[d]$, $y$ from $R[d]$
        if all turn $t$ in $H[d]$ has $t$.size == 0
            Remove all turns from $H[d]$, $R[d]$
    return $H$, $R$
```

Listing 4: Pseudocode of the algorithm selecting turns for evaluation under leave-one-domain-out setting

The seen metrics can be obtained similarly, using complementary logic – remove all slots not from a set of seen domains.

Appendix B Errors in Code Released by Zhao et al.

During the integration of the data preprocessing scripts released by Zhao et al. (2022) into the robust-DST codebase, two errors were discovered. Their occurrences and fixes were communicated to and confirmed by the original authors. Due to the particular configurations of experiments reported in the paper, the errors are unlikely to affect the published results. Nonetheless, their potential effects are discussed in this section.

B.1 SGD Multi-Frame Turns Error

When using the Python assignment operator with a variable referencing an object that is mutable on the right-hand side, the left-hand side variable is assigned the same reference. This is called a *shallow copy* – both variables now reference the same object, and modifications made on one variable will be reflected on the other variable. On the contrary, *deep copy* constructs a new object and recursively copies the content of the original object, so the new variable that is assigned to the deep-copied object is a separate entity from the original.

As mentioned in Section A.1.2, during the preprocessing of SGD data into D3ST format a Python `dataclass` called `TurnInfo` is used to store dialogue information such as the dialogue context and the prompt, on a per-frame basis. A new `TurnInfo` object is instantiated once per dialogue, and it is updated when iterating through each turn of the dialogue and each frame of that turn. In the released code, at the end of each frame a shallow copy of the current `TurnInfo` is added to a list holding `TurnInfo`s of the current turn. At the end of each turn, this temporary list is deep-copied and used to extend the output

This copying strategy will cause the contents in the `TurnInfo` of earlier frames to be replaced by that of the final frame, for turns with multiple frames. As a result, the processed dataset contains duplicated examples, while missing some examples for dialogue turns with multiple services. This error can be fixed by employing deep copy with appending to the temporary list at the end of each frame. With this change, using shallow copy when extending the output list at the end of each turn is sufficient. A pull request containing this fix was submitted[8].

The error may not have a large effect if both the training and test sets are processed this way. However, if the model is trained with this preprocessing error but evaluated on the correctly processed test set, the performance may be negatively affected.

B.2 MultiWOZ Block Domain Error

The processing script from Zhao et al. (2022) accepts an argument for domains to block. For each turn, if any of the slots with a non-empty value is from a domain in the set of blocked domains, the turn is skipped. However, the prompt is constructed from the ontology and domains active (*i.e.* domains with non-empty slot values) in the belief state directly without considering the blocked domains. The code behaves correctly if the option to only use active domains is set.

To be more specific, if not only using active domains, and assume there are five domains in the ontology with one domain being left-out. The processed dataset will only contain turns that do not have non-empty slots from the unseen domain (*i.e.* only the four domains

[8]`https://github.com/google-research/task-oriented-dialogue/pull/6`

may be present in the dialogue contexts), but each prompt will have descriptions of slots from all of the five domains.

This defeats the aim of cross-domain evaluation – to simulate an unseen service. Even though this error has no effect under the prompt format used by Zhao et al. (2022) (discussed in Section 4.2.2), under other training and decoding prompt formats, it significantly reduced the model performance on the left-out domain (shown in Appendix C).

The error can be fixed by simply removing the blocked domains from the set of domains in the ontology when generating the prompts.

Appendix C MultiWOZ Cross Domain Transfer Prompting Strategies

Other than the prompting strategy mentioned in Section 4.2.2, there are other logical prompting strategies when training and decoding on MultiWOZ under the leave-one-domain-out setting. The ones discussed in this section do not rely on the active domains in the belief states, and thus allow testing on turns not only containing the left-out domain.

Assume there are $n + 1$ domains in the training set, and the $(n + 1)$th domain is being left-out when training. $\langle \text{desc}_p^{(q)} \rangle$ denotes the index and description of the pth slot of domain q. If the 1st, nth, $(n + 1)$th domains have j, k, l slots respectively, we define the following three prompt formats:

- `A`: all slot descriptions from the seen domain

$$\langle \text{desc}_1^{(1)} \rangle \cdots \langle \text{desc}_j^{(1)} \rangle \cdots \langle \text{desc}_1^{(n)} \rangle \cdots \langle \text{desc}_k^{(n)} \rangle$$

- `B`: all slot descriptions from both the seen and unseen domains (*i.e.* the entire ontology)

$$\langle \text{desc}_1^{(1)} \rangle \cdots \langle \text{desc}_j^{(1)} \rangle \cdots \langle \text{desc}_1^{(n)} \rangle \cdots \langle \text{desc}_k^{(n)} \rangle \langle \text{desc}_1^{(n+1)} \rangle \cdots \langle \text{desc}_l^{(n+1)} \rangle$$

- `C`: only slot descriptions from the unseen domain

$$\langle \text{desc}_1^{(n+1)} \rangle \cdots \langle \text{desc}_l^{(n+1)} \rangle$$

In the MultiWOZ data augmentation experiment (Section 4.2.1), prompt format `A` is used for training and prompt `B` is used for decoding. Different to the setting in Section 4.2.2, each training and decoding example is prompted by a constant number of slot descriptions.

The results when leaving `attraction` domain out are shown in Table 14, where prompting formats with only descriptions of slots from the active domains in the belief state are denoted with the suffix "(active)".

Training Prompt Format	Decoding Prompt Format	JGA	JGA^{seen}	JGA^{unseen}
`A` (active)	`C` (active)	-	-	79.2
`B`	`B`	48.7	65.4	1.1
`A`	`B`	35.2	44.4	19.7
`A`	`C`	-	-	46.3

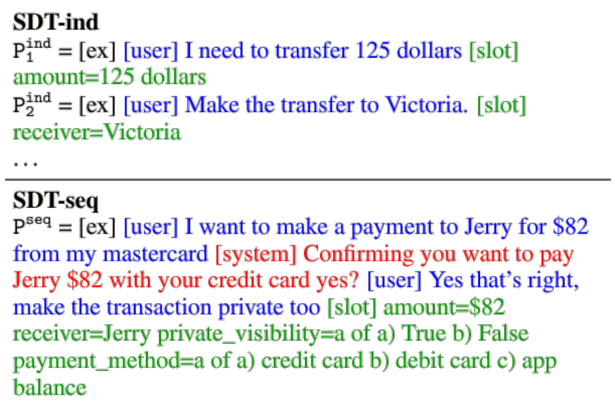
Table 14: MultiWOZ cross-domain results for different prompting strategies, when leaving the `attraction` domain out. Three-run averages.

The error in code by Zhao et al. (2022) (as mentioned in Section B.2) effectively prompts the model with prompt format `B` during training when not using active domains only. This corresponds to Row 2 of Table 14. The very low JGA^{unseen} is because the model learnt to ignore slots from the left-out domain, as their descriptions are in the prompts but they never exist in the target state during training.

When the model is trained using prompt format `A`, and decoded with prompt format `B` (Row 3), *i.e.* when the error is fixed. The JGA^{unseen} is increased, due to fewer empty predictions for slots from the unseen domain. Interestingly, the seen performance is reduced, indicating the D3ST model is very sensitive to prompt formats. When decoding with prompt format `C` (Row 4), the JGA^{unseen} is increased further, likely due to the prompts being more effective in prompting for predictions of slots from the unseen domain specifically.

When using only active domains (Row 1), the highest JGA^{unseen} is achieved. This is because,qp w under this setting, the test set only contains turns with slots exclusively from the unseen domain. Even though as mentioned in Section 3.3.3, only predictions for slots from the unseen domain are considered when calculating JGA^{unseen}, turns with slots from both the unseen domain and other domains are also included when not using the active setting. The better alignment between the prompts and the dialogue contexts when using `C` (active) may be a reason for the higher JGA^{unseen}.

Appendix D Other DST Models



SDT-ind
P_1^{ind} = [ex] [user] I need to transfer 125 dollars [slot] amount=125 dollars
P_2^{ind} = [ex] [user] Make the transfer to Victoria. [slot] receiver=Victoria
...

SDT-seq
P^{seq} = [ex] [user] I want to make a payment to Jerry for $82 from my mastercard [system] Confirming you want to pay Jerry $82 with your credit card yes? [user] Yes that's right, make the transaction private too [slot] amount=$82 receiver=Jerry private_visibility=a of a) True b) False payment_method=a of a) credit card b) debit card c) app balance

Figure 9: Illustration of prompt formats of SDT-ind and SDT-seq models (Gupta et al., 2022)