ShopTalk: A System for Conversational Faceted Search

Gurmeet Manku, James Lee-Thorp, Bhargav Kanagal, Joshua Ainslie, Jingchen Feng, Zach Pearson, Ebenezer Anjorin, Sudeep Gandhe, Ilya Eckstein Jim Rosswog, Sumit Sanghai, Michael Pohl, Larry Adams, D. Sivakumar
Google Inc, USA
{manku,jamesleethorp,bhargav,jainslie,jingchenfeng}@google.com
{zpearson,eanjorin,srgandhe,ilyaeck,jrosswog,sumitsanghai,mpohl,leadams,siva}@google.com

ABSTRACT

ShopTalk is a multi-turn conversational search system for online shopping. It handles giant and complex schemas that go beyond the state of the art slot-filling faceted search based systems. ShopTalk's dialog understanding system consists of a deep-learned parsing module, which interprets user utterances, and a primarily rules-based dialog-state tracker (DST), which updates the dialog state and formulates search requests intended for the fulfillment engine. The interface between the two modules consists of a minimal set of domain-agnostic *intent operators*, which instruct the DST on how to manage the dialog state. ShopTalk was deployed in 2020 on Google Assistant for Shopping searches.

CCS CONCEPTS

- **Computing methodologies → Natural language processing**;
- **Information systems → Online shopping**.

KEYWORDS

conversation systems, neural networks, semantic parsing

ACM Reference Format:
Gurmeet Manku, James Lee-Thorp, Bhargav Kanagal, Joshua Ainslie, Jingchen Feng, Zach Pearson, Ebenezer Anjorin, Sudeep Gandhe, Ilya Eckstein Jim Rosswog, Sumit Sanghai, Michael Pohl, Larry Adams, D. Sivakumar. 2022. ShopTalk: A System for Conversational Faceted Search. In *Proceedings of SIGIR eCom '22*. ACM, New York, NY, USA, 9 pages.

1 INTRODUCTION

Faceted search is a classic paradigm used by companies like Amazon, eBay and Yelp for searching a collection of objects by offering two knobs to users: text queries and facet preferences.

Text queries are keyword based; for example, *"red Nike shoes."* In recent years, with the rise of mobile devices and breakthroughs in Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU), longer and more natural language queries have risen in prominence, for example, *"show me some red Nike shoes please."*

(1) *"Show me some Nike women's shoes please."*
(2) *"Do you have anything in red?"*
(3) *"How about pink?"*
(4) *"Actually, almost any color will do; just make sure it's not white."*
<scrolls through products on display>
(5) *"Hmmm... let's also see Adidas."*
(6) *"Okay, it doesn't have to be Adidas but I want ones that are good for running."*
(7) *"Something that protects my feet in heavy rain."*
(8) *"Do you have anything less than a hundred bucks?"*
(9) *"Anything even cheaper?"*
<clicks on a product>
(10) *"Size 9."*
<selects product >
(11) *"I want to buy some red socks too."*

Figure 1: An utterance sequence parsed by ShopTalk. Utterances (1) and (11) trigger product category switches. Utterances (1), (4) and (6) express multiple preferences in the same utterance. Utterance (4) expresses a negative preference. Utterances (8) and (9) showcase range-oriented preferences and nudges over a numeric attribute. Almost all utterances refer to tags but utterances (4) and (10) refer to facets Color and Size by their name. Utterance (7) highlights a long text span that maps to tag waterproof.

Facet preferences are predicates over object attributes such as Brand and Price. Facets are usually presented to a user using dropdowns, checkboxes, radio buttons, or text fields for range specification. For example, in a shopping schema, a facet called ProductCategory might allow users to choose between shoes and televisions.

In this paper, we outline ShopTalk, which enables chatbot-style faceted search over complex schema in an industry setting. The user drives a ShopTalk conversation by expressing a series of preferences. Figure 1 shows an example utterance sequence that ShopTalk supports. Our design assumes (a) a shopping schema with facets and tags, (b) a search backend to fulfill user requests from query and facet constraints, and (c) query logs for the existing, non-conversational faceted search experience.

1.1 Challenges

The challenges in conversational, faceted search for online shopping stem from the size and complexity of the schema and the diversity of user utterances.

The shopping schema is large: Web-scale shopping schemas contain thousands of categories, each with thousands of attributes. For example, the Google product taxonomy has over 6,000 product categories [14]. The AliExpress taxonomy contains thousands of product categories, with individual categories such as Sports and Entertainment containing over 8,900 unique attributes [40]. Such schemas are much larger than schemas tackled by state-of-the-art slot filling dialog systems [5, 11, 12]. For comparison, the MultiWOZ [4, 27] and DSTC8 [28] task-oriented benchmarks consist of only thousands of attributes across all categories in the schema.

The shopping schema is complex: Facets may be boolean (e.g., WATERPROOF), numeric (e.g., PRICE: \$20 to \$200), discrete ordered (e.g., SIZE: XS, S, M, L, XL, XXL) or unordered (e.g., TYPEOFCAMERA: DISPOSABLE, POINT & SHOOT, SLR, MEDIUM FORMAT). Users specify preferences over these types in a variety of ways like *"it must be waterproof"*, *"i'm not looking for a throwaway camera"*, or *"show me something cheaper"*.

Some facets have multiple types. For example, the SIZE facet for SHOES may be both numeric (6, 7, 8, . . . , 14) and non-numeric (S, M, L, XL) at the same time. Also, a numeric tag like 5 may represent the SIZE or VOLUME or LENGTH.

The shopping schema is incomplete: Web-scale shopping schemas are dynamic, being constantly updated as new products, tags and facets are introduced. Also, shopping schemas are rich with respect to tags and facets for popular product categories (the "head" of the schema) but impoverished for less popular categories (the "tail" of the schema).

The ungrounded span problem: An ungrounded span in a user utterance is a token sequence referring to a tag that is missing in the schema. Examples:

(1) *"show me women's shoes without any ankle straps"*
(2) *"i want to buy lemon-scented hand soap"* → *"no, i actually prefer lavender"*

where *"ankle strap"*, *"lemon-scented"* and *"lavender"* refer to tags missing in the schema. In example (2), a smart dialog system would replace the preference for *"lemon-scented"* by *"lavender"* even though both are missing in the schema.

Synonymization problem: Users often conceptualize facets and tags differently from the vocabulary chosen by product manufacturers. For example, both *"does not get wet"* and *"protects my feet in rain"* are synonyms for tag WATERPROOF.

Utterances are contextual: Parsing an utterance may need context established by earlier utterances. Consider *"i want to buy a tv"* → *"something with 5 ports"* → *"can we increase that?"* The second utterance refers to a product category (TELEVISIONS) specified in the first utterance. The third utterance nudges a numeric facet specified in the second utterance.

Negative preferences: *"I don't want Nike"*, or *"I hate red"*, or *"don't show me front-loading dishwashers."* Negative preferences are usually unsupported by classic faceted search interfaces; they are common in spoken language interfaces.

Multiple preferences in an utterance: *"show me red Nike shoes"* specifies (PRODUCTCATEGORY = SHOES) AND (BRAND = NIKE) AND (COLOR = RED). A follow-on utterance like *"don't show me blue; I like red"* specifies two preferences: NOT (COLOR = BLUE) AND (COLOR = RED).

1.2 Contributions

We tackle the aforementioned challenges, as follows.

Modular system design: As highlighted in Section 1.1, building an end-to-end system over a web-scale shopping schema is challenging for slot filling systems because the number of slots is potentially in tens of millions. Training such as system would require collecting an enormous amount of training data. We tackled this problem by identifying a small number of *intent operators* – a succint representation of dialog state to dialog state transitions. This design choice simplifies the Parser: it only has to classify the latest utterance into a small set of intent operators. Our system has a 3-stage pipeline:

PARSE: A seq2seq based Parsing module for parsing user utterances into human-understandable *intent operators*; see Section 4.

MERGE: A Dialog State Tracker (DST) module for merging intents from all utterances to derive cumulative dialog state; see Section 5.

FULFILL: A Fulfillment module to respond to user utterances by displaying products that match a text query and cumulative facet preferences, both of which are derived from the dialog state; see Section 3.5.

Support for complex utterances & preferences: Our Parser & DST modules support a variety of facet types, negative preferences and multiple preferences per utterance. We also address the synonymization problem by successfully mapping spans like "water resistant" to WATERPROOF tag.

Ungrounded spans: Our Parser & DST modules tackle two problems arising from shopping schema incompleteness:

(1) We developed techniques to train the parser module to also recognize ungrounded spans from user utterances, along with the intent (see Section 4.3 for details).

(2) We enhanced the DST with a learned submodule to merge user preferences associated with ungrounded spans (which are not part of the schema) with cumulative user preferences so far. See Section 5.3 for details.

Domain agnostic design: A unique feature of our design is that the Parser sends domain-agnostic *intent operators* to the DST to update dialog state (see Section 3), which enables the design to work for multiple domains / verticals.

Real world deployment: ShopTalk was deployed with Google Assistant in March 2020, leading to a nearly ten fold increase in the number of follow-on shopping queries from users and a 52% increase in shopping dialog lengths.

2 RELATED WORK

Several approaches for task-oriented dialog state tracking have emerged over recent years [5, 11–13, 17, 22, 25, 26, 29, 32, 35, 41–43]; see also Weld et al. [36] for a comprehensive survey. Many companies, including Amazon [20, 39], Baidu [3] and Facebook [6, 21], are actively developing conversational e-commerce systems. Of the aforementioned studies, our work is perhaps most similar to that of Yang et al. [43], who pose the problem of dialog state tracking as a query tracking problem using an end-to-end text based attention model — after each user utterance, the system updates a single query for the fulfillment backend. We find that representing the user's preferences as a single query limits the user's visibility into the system's conversational understanding and prevents integrating practical dialog structure, such as a recency bias to selectively fulfill user preferences, which turn out to be critical in multi-turn conversations. Our design separates intent parsing from dialog state updates.

In contrast, end-to-end models perform perform parsing and state updates simultaneously. However, as a result they typically require large amounts of diverse training data to handle an exponential number of transitions, many of which are complex (e.g. involving synonyms). Recent improvements in large language models [33] have shown that by scaling the model size and pre-training on giant amounts of web data, end-to-end dialog models are able to handle a wide variety of dialog transitions; although it is currently unclear how to serve these large models in low-latency production systems. By splitting the parsing component and the dialog state tracker, we are able to train a lightweight parser model with a much smaller output vocabulary of dialog "transitions". With more direct control of the dialog state tracker, we are also able to ensure that prior user preferences are not dropped, that the final dialog state is grounded (factually correct) and more directly debug our system in production services.

Most existing task-oriented dialog state tracking systems assume a fixed vocabulary of tags and facets. Gao et al. [11] attempt to overcome the limitations of fixed-vocabularies by adopting a neural reading comprehension approach. They tackle the dialog state tracking problem by making three sequential decisions based on the existing dialog state and the incoming utterance: (i) Should this slot from the existing dialog state be carried over the new dialog state? (ii) What slot type should the model update? (iii) What slot value should be copied to the selected slot type? This sequential or gated decision making setup is a relatively common and successful approach; see also [38], for example. Gao et al. [11] base their model on BERT and use the open benchmark MultiWOZ dataset [4, 27]. However, their "zero-shot" attribute results are necessarily limited by the relative size of the MultiWOZ multi-turn dialog, which has only 24 facets and 4,510 tags.

Several multi-turn dialog datasets have emerged in recent years based on a Wizard of Oz collection setup [10, 37]. These datasets typically span multiple domains and are geared towards general virtual assistant settings. Nevertheless, even the largest of these datasets (at the time of writing), namely the Schema-Guided Dialogue dataset [28], contains significantly fewer facets (214) and tags (14,139) than ours. In contrast, a web-scale shopping schema such as the Google product taxonomy has over 6,000 product categories [14]. The AliExpress taxonomy has thousands of product categories, with a single category – Sports & Entertainment – holding over 8,900 unique attributes [40], yielding a schema with potentially tens of millions of attributes. Moreover, such schemas are understood to be incomplete and dynamic; new tags and facets are introduced almost daily.

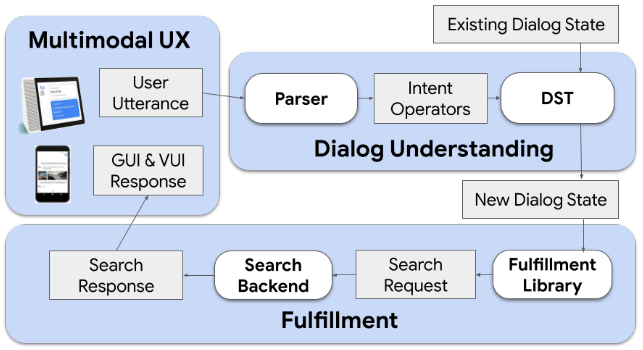

Figure 2: ShopTalk system architecture.

3 SYSTEM OVERVIEW

Figure 2 shows a block diagram for ShopTalk. The Parser parses a user utterance into intent operators (defined in Section 3.2). The DST uses these intent operators to update the existing dialog state to produce a new dialog state (defined in Section 3.1). The fulfillment module computes a query and facet constraints from the new dialog state. Products retrieved by the fulfillment module are shown to the user.

3.1 Dialog State Representation

We chose a structured format: an unordered set of predicates over facets ("slots") and tags ("values") to represent cumulative search preferences specified by a user so far. Some of the tags in dialog state may be ungrounded spans. An example:

$$\begin{aligned}&(\textsc{ProductCategory} = \textsc{shoes})\\ &\textsc{and } ((\textsc{Size} = 10) \textsc{ or } (\textsc{Size} = 11))\\ &\textsc{and } (\textsc{Color} \neq \textsc{red}) \textsc{ and } (\textsc{Color} \neq \textsc{blue})\\ &\textsc{and } \textit{"square heels"}\\ &\textsc{and } \textsc{SortOrder} = (\textsc{Price}, \textsc{ascending})\end{aligned}$$

Maintaining dialog state in a structured format, as opposed to just a query summarizing search preferences, has two primary benefits: (1) The dialog state may be echoed back to the users in a human friendly format for *grounding*, which helps the user establish trust that their utterance was understood by the system. (2) The dialog state is portable and domain agnostic. Any specific application only needs an adapter module to convert dialog state into their fulfillment language. For shopping, we convert dialog state to a combination of search query and facet restricts.

3.2 Intent Operators

The Parser module parses user utterances into intents (see Figure 3). An intent has an intent operator and its associated arguments. Intent operators may be seen as DST operators – they instruct the DST on how to update the existing dialog state. We designed intent operators carefully as a finite, minimal and complete basis for all possible DST $\rightarrow$ DST transitions. Intent operators act on either facets or tags.

The are two **tag-level intent operators** which operate on specific tags:

(1) SetValueOp($\langle$tag$\rangle$, $\langle$predicate-type$\rangle$, $\langle$inclusivity$\rangle$) is the most common intent operator in real world dialog. It represents utterances such as *"show me size 8"*, *"show me size 8 also"*, *"show me size 8 only"* and *"size 8 or more"*. Argument $\langle$inclusivity$\rangle$ has 3 possible values: inclusive (*"also"*), exclusive (*"only"*) and undefined (unspecified). Argument $\langle$predicate-type$\rangle$ has 6 possible values: equals, not_equals, less_than, less_eq, greater_than and greater_eq to represent $=$, $\neq$, $<$, $\leq$, $>$ and $\geq$, respectively. Predicate types for $<$, $\leq$, $>$ and $\geq$ express ranges which are valid only for facets of type numeric and categorical ordered. If $\langle$predicate-type$\rangle$ expresses ranges, the dialog state update requires merging of tag ranges. Otherwise, a dialog state update amounts to simply appending ($\langle$tag$\rangle$, $\langle$predicate-type$\rangle$) to dialog state. Additionally, if $\langle$inclusivity$\rangle$ equals exclusive, then all other tags in the same facet as $\langle$tag$\rangle$ are cleared.

(2) ClearValueOp($\langle$tag$\rangle$) represents utterances such as *"i don't care if it's red or not"*; the DST forgets $\langle$tag$\rangle$ by clearing it out. Note that this is different from *"i don't want red"*, which would be SetValueOp(red, not_equals).

Facet-level intent operators trigger facet-level adjustments:

(1) ClearFacetOp($\langle$facet$\rangle$) clears all tags and predicates for $\langle$facet$\rangle$, e.g., *"any color will do"* clears out any predicates associated with facet Color.

(2) ClearAllFacetsOp() clears dialog state by removing all tags and predicates for all facets, e.g., *"let's start over"*.

(3) NudgeFacetOp($\langle$facet$\rangle$, $\langle$nudge-direction$\rangle$) where $\langle$nudge-direction$\rangle$ may be positive or negative adjusts the tag for an ordered facet up or down, e.g., *"i want something larger"*.

(4) OrderByOp($\langle$facet$\rangle$, $\langle$sort-direction$\rangle$) specifies the sort order for fulfillment, e.g., *"show me the cheapest"*.

Some user utterances specify multiple intent operators. For example, *"i don't care about the color but i want size 10"* parses into two intent operators. We also emphasize that the intent operators defined above do not incorporate any shopping specific terminology and are therefore domain agnostic.

User utterance	Intents (Output of Parser)
(1) *"Show me some Nike shoes"*	SetValueOp(equals, nike)
(2) *"Something for running"*	SetValueOp(equals, running)
(3) *"Adidas ones too please"*	SetValueOp(equals, inclusive, adidas)
(4) *"Orange is okay but I don't want pink"*	{SetValueOp(equals, orange); SetValueOp(not_equals, pink)}
(5) *"Do you have anything in razmatazz?"*	SetValueOp(equals, *"razmatazz"*)
(6) *"Actually any color is OK"*	ClearFacetOp(Color)
(7) *"Size 9"*	SetValueOp(equals, size9)
(8) *"Show me something bigger"*	NudgeFacetOp(increase, Size)
(9) *"It doesn't have to be black"*	ClearValueOp(black)
(10) *"Do you have anything less than fifty bucks?"*	SetValueOp(less_than, price50)
(11) *"start over"*	ClearAllFacetsOp()

Figure 3: Examples of user utterances mapped to intents. In utterance (5), *"razmatazz"* is an ungrounded span. Utterance (8) needs the context of utterance (7) for interpretation.

3.3 Intent Parsing & Dialog State Tracking

Decomposition of dialog understanding into intent parsing (Parser) and dialog state tracking (DST) is motivated by the scale and complexity of the shopping schema. The number of possible transitions from old dialog state to new dialog state is potentially of quadratic complexity in the number of unique tags in the schema. Factoring dialog state transitions into a small-sized ($O(1)$) set of intent operators simplifies training data collection dramatically. Instead of collecting training examples for all possible dialog state transitions, we just need a representative sample for each intent operator. For example, rather than collect training data examples of the form *"i want red only"*, *"i want blue only"*, *"i want Nike only"*, . . . , it suffices to identify an intent operator for the canonical utterance *"i want $\langle$tag$\rangle$ only"* and then collect training examples for a sample of $\langle$tag$\rangle$ values. Note that canonical utterances are both product category and domain agnostic.

3.4 Annotated Spans vs Ungrounded Spans

A span is a token sequence in an utterance. A span can be annotated if it is recognized by the schema as a tag or a facet. Such annotations constitute a strong signal that the span is relevant, and that the Parser should copy the span to construct an intent operator; see Section 4. For the DST, updating the dialog state for an intent operator with an annotated span reduces to a facet (slot) filling problem; see Section 5. For example, if we know from our schema that red is a Color, then the DST knows to fill in the Color slot with red.

A web-scale shopping schema is inherently incomplete. Many spans cannot be mapped to tags or facets in the schema. Such *ungrounded spans* are more difficult for the Parser to detect, and also cannot be mapped directly to facets by the DST. The ungrounded span problem requires special care in both the Parser and DST; see Sections 4 and 5, respectively.

3.5 Fulfillment

The Fulfillment module converts dialog state into a combination of search query and facet restricts for a faceted search engine like Apache Solr on Lucene [1, 2]. Facet restricts are easy to compute from facet predicates. We construct the search query by concatenating a human-readable category name with all ungrounded spans and their associated predicates. The search query and facet restricts are sent to a shopping backend to retrieve matching products.

4 THE PARSER MODULE

The Parser module is the primary learned component of ShopTalk. The Parser parses a user utterance into a set of intents, based on the dialog context.

Parsing utterances into intents is known as semantic parsing [19], for which several modeling approaches have been proposed [8, 9, 18, 19, 30, 31]. We formulated our parsing problem as a sequence-to-sequence task, inspired by the success of such models [8, 9, 23, 44].

Examples of input-output training pairs for the Parser are listed in Figure 3. Note that the Parser outputs a *sequence of intents* to accommodate multiplicity of user preferences — see utterance (4) in Figure 3, for example. Also, context is important; in utterance (2) in Figure 3, the user simply specifies *"running"* without specifying that PRODUCTCATEGORY is SHOES, which needs to be inferred from the first utterance.

For ShopTalk, there were two primary challenges to training an effective Parser model. Firstly, for web-scale schemas [14, 40] with millions of unique tags, the number of unique intents is at least O(100M) due to combinatorial explosion: Our intents are parameterized by 6 intent operator types, 8 predicate types, 3 inclusivity types, and over a million unique tag values (see Section 1). Thus collecting utterances for all possible intents is infeasible; utterances for multiple intents (e.g., utterance (4) in Figure 3) make it even more challenging. We overcome this difficulty by using a limited set of structured intent; see Section 3.2.

A second challenge pertains to ungrounded spans defined in Section 1. The Parser model needs to recognize such phrases and also predict the right intent operators (e.g., utterance (4) in Figure 3). Note that for our target search application, fulfillment is feasible even for ungrounded spans, given access to a text search backend. Dealing with ungrounded spans also leads to challenges with dialog state tracking which we discuss in Section 5.

4.1 Training Data

We develop three strategies to collect training data.

Dataset D1 from raters: Collecting utterances for every intent is infeasible for O(100M) intents. However, expressing preference for NIKE shoes is quite similar to expressing preferences for RUNNING shoes — see utterances (1) and (2) in Figure 3, both of which map to the same *intent signature*, i.e., an intent that is not yet bound to a specific facet or tag (SETVALUEOP EQUALS for this example). So we develop a strategy to collect utterances for every possible intent signature, for a small number of attributes. The total number of unique intent signatures is around 100 and is much more manageable. We develop techniques to generalize to other attributes (Section 4.2) and ungrounded spans (Section 4.3). We select 10 attributes from each of 6 categories (a mix of discrete, numeric and boolean attributes). For each resulting intent, we instruct raters to specify at least 5 utterances (with an emphasis on diversity). For example, given the intent SETVALUEOP(COLOR, NOT_EQUALS, RED) for shoes, we obtain example utterances such as *"I don't like red"*, *"don't show me red shoes"*, *"I hate red"*, etc. To handle the complexity of the synonymization problem for popular facets like PRICE, we collect additional utterances. For example, prices can be expressed quantitatively (e.g., *"under $50"*) or qualitatively (*"something cheap"*).

Grammar for _NegationUtterance

_NegationUtterance	→	(_IDontWant) (_Condition) | (_IWant) (_NotCondition)
_IDontWant	→	"i (_Dont) want" | "i (_Dont) like" | "i wouldn't like" | "i dislike" | "i hate" | "i (_Dont) want to see"
_IWant	→	"i want" | "i like" | "show me"
_Dont	→	"don't" | "do not"
_Condition	→	"⟨TAG⟩"
_NotCondition	→	"no ⟨TAG⟩"

Example utterances for ⟨tag⟩ = *"red"*: *"i don't want red"*, *"i do not want red"*, *"i hate red"*, *"i want no red"*, ...

Figure 4: Grammar-based training data generation.

Dataset D2 from query logs: Yang et al [43] observe that search engine logs are a large-scale source of user data for faceted search. However, two challenges emerge. First, queries are much less verbose than user utterances on the assistant; Second, we still need to annotate each query with the right intent. To leverage search engine logs for training, we first select frequently occurring queries, i.e., those that are queried at least 100 times and then filter to long product-seeking queries. To automatically annotate the intent for each query, we further choose two subsets of queries:

(1) Fully parsed queries: Labeling such queries with intents is easy because all terms therein belong to the shopping schema. Such queries often have multiple intents.

(2) Well understood regular expressions: these are queries that match a whitelist of carefully selected regular expressions. For example, using the regular expression *"$category that are good for $span"*, we can match queries such as *"shoes that are good for back pain"*. For such queries, we can construct the intent, namely SETVALUE(EQUALS, $span). This approach helps us construct training examples with unknown attributes for $span. We can also obtain negations using templates like *"$category without $span"* (e.g., *"shirts without stripes"*).

Since search is a single-turn experience, all the utterances that we obtain correspond to a SetValueOp. Using query logs, we obtained over 100K training examples for the model.

Dataset D3 from grammars: Context-free grammars can generate user utterances, as shown by Yang et al. [43]. To generate a realistic grammar, we leveraged the rater utterances from D1. For each intent signature, we collected applicable rater utterances, *delexicalized* the attribute name (*"show me red ones"* → *"show me $tag ones"*) and handcrafted a grammar to generate much of the delexicalized utterances; see Figure 4 for an example for SetValueOp not_equals.

Next, we instantiated the grammar to generate all possible templates (without binding to specific tag values) yielding around 500K. Then, we sample tag values from our schema to produce several million utterances for the Parser model. While rater utterances and user queries correspond to high training quality data, not all utterances obtained from grammars are fluent, therefore these were given a lower weight for training.

We got the best model quality by training with all three datasets D1, D2 and D3. The grammar-based dataset D3 gave a substantial boost to model performance, despite it being lower quality.

4.2 Contextual Annotations

Our training data contains only a small subset of attributes from the schema. For generalization to all attributes at inference time, we provide *hints* in the form of special tokens that identify known attributes in the utterance.

To compute annotations, we first predict the product category for the utterance with an existing classifier.[1] We then match all known tags (and their curated synonyms) in the product category to the query in order to determine the relevant annotations. Annotations are featurized using a graph encoding, as described below.

4.3 Model Architecture

We adopt the Transformer [34]-based encoder-decoder architecture outlined by Shaw et al. [30, 31], with support for handling *graph structured inputs* and a *copy mechanism* in the decoder. We also introduce a novel annotation dropout scheme to improve generalization for ungrounded spans.

The input to the transformer encoder is the user utterance along with its contextual annotations. The output from the decoder is the intents. The input text is tokenized into wordpieces [7]. The annotations are treated as a special token and featurized purely by their type. The names of the intent operators and their parameters are added to the decoder vocab.

Graph inputs: Shaw et al. [30, 31] represent the input utterance and corresponding annotations using a graph whose nodes correspond to either (a) tokenized wordpieces in the input or (b) the annotations. Graph edges associate an annotation with the corresponding text spans. All edges are parameterized with key and value embeddings that are then used in the self-attention formulation; see [30, 31] for details.

[1] If the product category is unclear, as it often is for follow-on utterances in which users do not repeat the product category, we reuse the category from a previous utterance in the dialog. The classifier may also detect a new product category, resulting in a category switch; see Section 6.

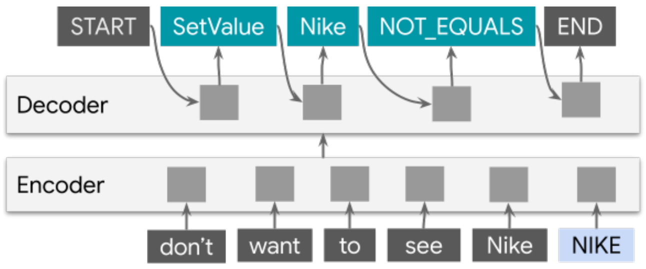

Figure 5: The Parser model. Input: user utterance — tokenized and annotated with known attributes (e.g., NIKE in the blue box). Output: intent shown on top. The text token 'Nike' is copied from input to output.

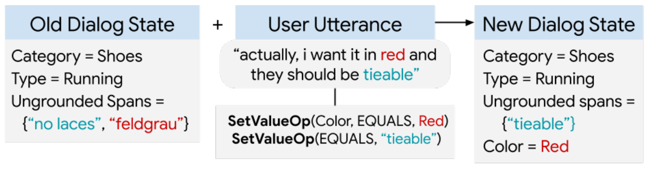

Figure 6: Exemplary dialog state update.

Copy mechanism: This allows the decoder to copy tokens directly from the input (in addition to generating tokens from the vocab). This is useful in our application for copying over attribute phrases.

Annotation dropout: Our model copies text spans, not annotations. This helps the model generalize to unrecognized attributes with zero annotations ("ungrounded text spans"). To further improve generalization, we also randomly drop annotations from the input for a large fraction of examples. This encourages the model to copy over text spans that are not necessarily annotated.

Low latency: We chose a small transformer with 2 encoder layers and 1 decoder layer (see Figure 5). We observed only a limited drop in quality when compared with a full transformer.

5 DIALOG STATE TRACKER (DST)

The Dialog State Tracker (DST) manages dialog state. Given an existing dialog state and a set of intents predicted by the Parser for the latest utterance, the DST computes the new dialog state. An example of a DST update using intent operators is shown in Figure 6.

5.1 Intents With Annotated Spans

If all tags and facets in an intent are recognized by the shopping schema, then updating the dialog state reduces to a slot filling problem. For example, if we know that span *"red"* maps to tag RED in facet COLOR, then the DST can infer the following exemplary rules:

(1) *"i only want red"* → SETVALUEOP(COLOR, EQUALS, RED, EXCLUSIVE) ⇒ clear COLOR facet, then append tag RED

(2) *"i would also like to see red ones"* → SETVALUEOP(COLOR, EQUALS, RED, INCLUSIVE) ⇒ append RED to facet COLOR.

(3) *"just make sure it's not blue"* → SETVALUEOP(COLOR, NOT_EQUALS, BLUE) ⇒ append NOT BLUE to facet COLOR.

Thus DST update rules vary depending on the intent operator and its arguments (⟨facet⟩, ⟨tag⟩, ⟨inclusivity⟩, etc.).

The ordering of intents is important. Recall that the Parser outputs a *sequence of intents*, the order of which is determined by the user utterance. However, examples below illustrate that this order may be inappropriate for dialog state updates.

EXAMPLE 1: *"just make sure they are not blue, but reset other color preferences"* may parse into the sequence {SETVALUEOP(COLOR, NOT_EQUALS, BLUE), CLEARFACETOP(COLOR)} for which the correct ordering is CLEARFACETOP(...) followed by SETVALUEOP(...).

EXAMPLE 2: *"make sure it's under* \$100 *but reset all other preferences"* may parse into the sequence {SETVALUEOP(PRICE, LESS_THAN, \$100), CLEARALLFACETSOP()}. The correct ordering is CLEARALLFACETSOP() followed by SETVALUEOP(PRICE, LESS_THAN, \$100)

Examples like those above led us to develop a hand-crafted DST update algorithm in which clear operations are prioritzed; see Figure 7.

Step 1: Apply any facet independent intent operators (CLEARALLFACETSOP, ORDERBYOP).

Step 2: Group remaining intent operators (*VALUEOP, *FACETOP) by facet.

Step 3: Within each facet grouping, reorder the intent operators such that clear operations appear first.

Step 4: For each SETVALUEOP, clear any existing predicates which conflict with it.

Step 5: Apply remaining intent operators sequentially.

Figure 7: DST update algorithm.

5.2 Intents With Ungrounded Spans

Ungrounded spans correspond to attributes missing in the schema. Ungrounded spans in intents cannot be mapped directly to tags or facets. For an utterance like *"i want shoes that are tieable"*, the dialog state update needs an understanding of the relationship between the ungrounded span *"tieable"* and various tags and facets already present in dialog state. Note that the existing dialog state itself may contain one or more ungrounded spans as illustrated in Figure 6.

To handle ungrounded spans, we developed a multitask BERT-based deep learned classifier (described in Section 5.3) which answers three specific questions in the context of the product category:

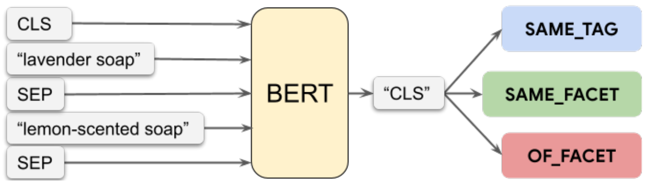

Figure 8: Classifier model multitask architecture.

(1) `SAME_TAG(tag1, tag2)` – do both tags represent the same concept or attribute? (e.g. *"sleeveless dresses"* and *"dresses without sleeves"*)
(2) `SAME_FACET(tag1, tag2)` – do both tags belong to the same facet? (e.g *"turquoise shoes"* and *"purple shoes"*)
(3) `OF_FACET(tag, facet)` – does tag belong to facet? (e.g tag = *"turquoise shoes"* and facet = *"shoe color"*)

These three questions are context dependent. For example, an *"ivory dress"* and a *"white dress"* belong to the same facet (*"color"*), whereas an *"ivory figurine"* and *"white figurine"* do not (*"material"* vs *"color"*).

For a given DST update, we first identify tag and facet relations by performing pairwise comparisons of every ungrounded span in an intent with every tag, facet and ungrounded span in existing dialog state. We then apply DST update algorithm in Figure 7.

5.3 Ungrounded Span Classifier Model

Figure 8 shows the multitask classifier model. We use a dense projection layer with multiple tasks on top of the BERT output "CLS" token. During training, all tasks are co-trained and all share the same encoder weights, but each task has its own (learned) projection function. For inference, each projection outputs a probability that the input pair of attributes satisfies each relationship.

The classifier is trained on examples of the form `attribute1`, `attribute2` ↦ `relationships`, where relationships is a subset of {`SAME_FACET`, `SAME_TAG`, `OF_FACET`}.[2] We source gold training examples directly from the schema. For example, because we know that both *"red"* and *"blue"* are colors, we can construct three positive examples: (*"red shoes"*, *"blue shoes"*) ↦ {`SAME_FACET`}, (*"red shoes"*, *"shoe color"*) ↦ {`OF_FACET`}, (*"blue shoes"*, *"shoes color"*) ↦ {`OF_FACET`}. Negative examples can be formed by sampling tags from different facets in the schema.

We also generate silver training examples from three sources:

(1) Category Builder [24] performs semantic set expansions like {Ford, Nixon} → {Obama, Bush, Regan, Trump, ...} and {Ford, Chevrolet} → {Jeep, GMC, Tesla, ...}. Starting with seed sets sampled from the shopping schema, e.g. {*"red"*, *"blue"*}, we construct `SAME_FACET` examples using Category Builder; e.g., {*"red"*, *"blue"*} ↦ {*"turquoise"*, *"feldgrau"*, ...}.

[2]Two queries can satisfy multiple labels. For example, any two attributes that satisfy `SAME_TAG`, also satisfy `SAME_FACET` by definition.

(2) Using an internal graph of "Is-A" relationships (e.g. *"red"* is a color), constructed using Hearst patterns [16], we can infer SAME_FACET relationships by checking whether two tags share several parent node (e.g., *"red"* and *"blue"* both satisfy "Is-A" color and "Is-A" primary color). We exclude common word parents such as "word" and "thing".
(3) Internal web synonym services similar to He et al. [15] generate SAME_TAG examples like *"big shoes"* and *"large shoes"*.

Model quality was further improved by jointly pre-training the encoder on three tasks: (1) masked language modelling (MLM), (2) predicting query parse trees and (3) a general synonym task.

6 DEPLOYMENT

The ShopTalk Parser & DST modules enable a conversational shopping experience for any shopping website or app that builds upon faceted search, for example, using Apache Solr on Lucene [1, 2]. We launched ShopTalk with Google Assistant to leverage pre-existing infrastructure for dialog-based systems.

In virtual assistants like Google Assistant, Alexa, Siri, Cortana, Dueros (Baidu) and AliGenie, multiplexing between backends like shopping, maps, weather and search for a coherent user experience is challenging. For example, which backend should fulfill *"I want to buy some coffee"* – maps or shopping? Another example: *"i want to buy shoes"* → *"size 9"* → *"waterproof please"* → *"is it raining?"* The last utterance should be fulfilled by the Weather backend, yet retain shopping context. For initial deployment, we relied on heuristics with a plan for a more robust solution in future.

For deployment with Google Assistant, we introduced system prompts like *"anything else?"* and *"got it! What kind of shoe did you have in mind?"* to encourage users to utter their preferences. These prompts led to new dialog acts of the forms *"no"*, *"not now"*, *"not really"*, *"no thank you"*, ... or *"yes"*, *"yeah"*, ..., which in turn required some straightforward extensions to the Parser & DST modules for parsing.

7 RESULTS

Pre launch metrics: For an offline, end-to-end system evaluation, we asked 500 domain experts, human raters to interact with a limited deployment of ShopTalk. Raters were instructed to search for products using voice commands, mimicking the behavior of online shoppers. Their first utterance would specify a broad product category, e.g., *""i want to buy* ⟨ProductCategory⟩ *"*. In follow-on utterances, raters were asked to narrow down the set of products using refinement utterances like *"don't show me pink"*. At every turn, the assistant would show an updated list of products. Raters were instructed to continue the conversation until they found a product matching their needs, or until the conversation broke down for any reason. At the end, raters provided a satisfaction rating for the full conversation. Over 70% raters reported high satisfaction scores.

Post launch metrics: Following deployment of ShopTalk online on Assistant, we observed the following. Please note that we are only able to share relative metrics owing to proprietary considerations.

(1) The number of follow-on shopping queries increased 9.5x fold which suggests that users started refining their queries, engaging in a dialog with the assistant.

(2) Average dialog length (number of turns) increased by 52% , highlighting the improved capability of parsing (Parser) and dialog state tracking (DST).

(3) User engagement with displayed products (for example, to inspect detailed product information) in response to their utterances also increased by 52%.

(4) In line with our pre-launch rater study, over 70% of the raters using the production system had high user satisfaction score. Despite dialog length increase, user satisfaction over the baseline system increased.

(5) Finally, we found that roughly a third of user utterances contained at least one ungrounded span, highlighting how incomplete schemas can be.

8 LESSONS LEARNED & FUTURE WORK

ShopTalk deployment on Google Assistant showcased how a real world faceted search system can be conversationalized. The underlying schemas for such systems are web-scale (large, complex, incomplete and dynamic), which necessitated clever system design and training data collection techniques. However, further work is needed for a richer experience:

Support for questions and answers: Many users ask questions on the search result page, e.g., questions about facets for specific products (*"does it have a usb charging port?"* or *"is it compatible with Xbox?"*) or about comparing products (*"which phone has the longest battery life?"*).

Better explanation of results: Further work is needed for (a) annotating every product displayed to the user by justifications, and (b) explaining the set of products as a whole succinctly. For example, users are often surprised if the result set became zero, or remained the same as before; in both cases, an explanation would improve user experience.

Out of scope utterances: Many times, user utterances correspond to functionality that we currently don't support, e.g., comparison of two products. Presently, the Parser and DST fail to parse such utterances. A more graceful handling of such utterances would improve user experience.

Preference elicitation: To encourage users to specify their preferences, our system generates a simple 'refinement prompt' like *"anything else?"* or *"got it! what kind of* ⟨productcategory⟩ *did you have in mind?"* This resulted in a wide variety of responses like *"i don't know"*, *"stop asking me for specifics"*, *"give me a second"*, ... Further work is needed for nuanced and context-aware refinement prompts and responses for preference elicitation.

REFERENCES

[1] Apache Software Foundation. [n.d.]. Apache Lucene. https://lucene.apache.org/. [Online; accessed 3-February-2021].

[2] Apache Software Foundation. [n.d.]. Apache Solr. https://lucene.apache.org/solr/. [Online; accessed 3-February-2021].

[3] Baidu NLP, Baidu Inc. 2019. Conversational Recommendation. https://ai.baidu.com/file/3DE0BE74852246079C2A3AE835397002. [Online; accessed 1-July-2022].

[4] Paweł Budzianowski, Tsung-Hsien Wen, et al. 2018. MultiWOZ - A Large-Scale Multi-Domain Wizard-of-Oz Dataset for Task-Oriented Dialogue Modelling. In *EMNLP 2018*.

[5] Guan-Lin Chao and Ian Lane. 2019. BERT-DST: Scalable End-to-End Dialogue State Tracking with Bidirectional Encoder Representations from Transformer. In *INTERSPEECH*.

[6] Paul A. Crook, Shivani Poddar, et al. 2019. SIMMC: Situated Interactive Multi-Modal Conversational Data Collection And Evaluation Platform. https://arxiv.org/abs/1911.02690

[7] Jacob Devlin, Ming-Wei Chang, et al. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL 2019*.

[8] Li Dong and Mirella Lapata. 2016. Language to Logical Form with Neural Attention. In *ACL 2016*.

[9] Li Dong and Mirella Lapata. 2018. Coarse-to-Fine Decoding for Neural Semantic Parsing. In *ACL 2018*.

[10] Layla El Asri, Hannes Schulz, et al. 2017. Frames: a corpus for adding memory to goal-oriented dialogue systems. In *SIGDIAL 2017*.

[11] Shuyang Gao, Abhishek Sethi, et al. 2019. Dialog State Tracking: A Neural Reading Comprehension Approach. In *SIGDIAL 2019*.

[12] Rahul Goel, Shachi Paul, et al. 2019. HyST: A Hybrid Approach for Flexible and Accurate Dialogue State Tracking. In *Proc. Interspeech 2019*.

[13] Yu Gong, Xusheng Luo, et al. 2019. Deep Cascade Multi-Task Learning for Slot Filling in Online Shopping Assistant. In *AAAI 2019*. 6465–6472.

[14] Google Inc. 2021. Google Product Taxonomy. https://www.google.com/basepages/producttype/taxonomy.en-US.txt. [Online; accessed 3-February-2021].

[15] Yeye He, Kaushik Chakrabarti, et al. 2016. Automatic Discovery of Attribute Synonyms Using Query Logs and Table Corpora *(WWW '16)*.

[16] Marti A. Hearst. 1992. Automatic Acquisition of Hyponyms from Large Text Corpora. In *COLING 1992 Volume 2: The 15th International Conference on Computational Linguistics*.

[17] Matthew Henderson, Blaise Thomson, et al. 2014. Word-Based Dialog State Tracking with Recurrent Neural Networks. In *SIGDIAL 2014*.

[18] Srinivasan Iyer, Ioannis Konstas, et al. 2017. Learning a Neural Semantic Parser from User Feedback. In *ACL 2017*.

[19] Robin Jia and Percy Liang. 2016. Data Recombination for Neural Semantic Parsing. In *ACL 2016*.

[20] Kommu, Jaya Prakash and Srinivasan, Sandeep. 2021. Build conversational experiences for retail order management using Amazon Lex. https://aws.amazon.com/blogs/machine-learning/build-conversational-experiences-for-retail-order-management-using-amazon-lex/. [Online; accessed 1-July-2022].

[21] Satwik Kottur, Seungwhan Moon, et al. 2021. SIMMC 2.0: A Task-oriented Dialog Dataset for Immersive Multimodal Conversations. https://arxiv.org/abs/2104.08667

[22] Sungjin Lee and Maxine Eskenazi. 2013. Recipe For Building Robust Spoken Dialog State Trackers: Dialog State Tracking Challenge System Description. In *SIGDIAL 2013*.

[23] Kevin Lin, Ben Bogin, et al. 2019. Grammar-based Neural Text-to-SQL Generation. arXiv:1905.13326 [cs.CL]

[24] Abhijit Mahabal, Dan Roth, et al. 2018. Robust Handling of Polysemy via Sparse Representations. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*.

[25] Nikola Mrkšić, Diarmuid Ó Séaghdha, et al. 2017. Neural Belief Tracker: Data-Driven Dialogue State Tracking. In *ACL 2017*.

[26] Julien Perez and Fei Liu. 2017. Dialog state tracking, a machine reading approach using Memory Network. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*.

[27] Osman Ramadan, Paweł Budzianowski, et al. 2018. Large-Scale Multi-Domain Belief Tracking with Knowledge Sharing. In *ACL 2018*.

[28] Abhinav Rastogi, Xiaoxue Zang, et al. 2020. Towards Scalable Multi-Domain Conversational Agents: The Schema-Guided Dialogue Dataset. *AAAI 2020* (2020).

[29] Liliang Ren, Kaige Xie, et al. 2018. Towards Universal Dialogue State Tracking. In *EMNLP 2018*.

[30] Peter Shaw, Philip Massey, et al. 2019. Generating Logical Forms from Graph Representations of Text and Entities. In *ACL 2019*.

[31] Peter Shaw, Jakob Uszkoreit, et al. 2018. Self-Attention with Relative Position Representations. In *NAACL 2018*.

[32] Blaise Thomson and Steve Young. 2010. Bayesian update of dialogue state: A POMDP framework for spoken dialogue systems. *Computer Speech and Language* 24, 4 (March 2010).

[33] Romal Thoppilan, Daniel De Freitas, et al. 2022. LaMDA: Language Models for Dialog Applications. *arXiv preprint arXiv:2201.08239* (2022).

[34] Ashish Vaswani, Noam Shazeer, et al. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, Vol. 30.

[35] Zhuoran Wang and Oliver Lemon. 2013. A Simple and Generic Belief Tracking Mechanism for the Dialog State Tracking Challenge: On the believability of observed information. In *SIGDIAL 2013*.

[36] H. Weld, X. Huang, et al. 2021. A survey of joint intent detection and slot-filling models in natural language understanding. arXiv:2101.08091 [cs.CL]

[37] Tsung-Hsien Wen, David Vandyke, et al. 2017. A Network-based End-to-End Trainable Task-oriented Dialogue System. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*.

[38] Chien-Sheng Wu, Andrea Madotto, et al. 2019. Transferable Multi-Domain State Generator for Task-Oriented Dialogue Systems. In *ACL 2019*.

[39] Liqiang Xiao, Jun Ma, et al. 2021. End-to-end conversational search for online shopping with utterance transfer. In *EMNLP 2021*. https://www.amazon.science/publications/end-to-end-conversational-search-for-online-shopping-with-utterance-transfer

[40] Huimin Xu, Wenting Wang, et al. 2019. Scaling up Open Tagging from Tens to Thousands: Comprehension Empowered Attribute Value Extraction from Product Title. In *ACL 2019*.

[41] Puyang Xu and Qi Hu. 2018. An End-to-end Approach for Handling Unknown Slot Values in Dialogue State Tracking. In *ACL 2018*.

[42] Zhao Yan, Nan Duan, et al. 2017. Building Task-Oriented Dialogue Systems for Online Shopping. In *AAAI'17*. 4618–4625.

[43] Yunlun Yang, Yu Gong, et al. 2018. Query Tracking for E-commerce Conversational Search: A Machine Comprehension Perspective. arXiv:1810.03274 [cs.CL]

[44] Rui Zhang, Tao Yu, et al. 2019. Editing-Based SQL Query Generation for Cross-Domain Context-Dependent Questions. In *EMNLP-IJCNLP 2019*.