Guided K-best Selection for Semantic Parsing Annotation

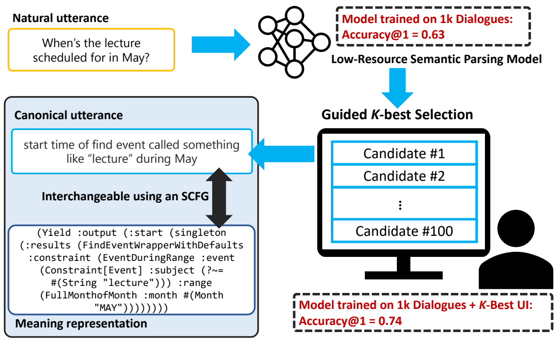
Anton Belyy[*1], **Chieh-Yang Huang**[*2], **Jacob Andreas**[3],
Emmanouil Antonios Platanios[3], **Sam Thomson**[3], **Richard Shin**[3],
Subhro Roy[3], **Aleksandr Nisnevich**[3], **Charles Chen**[3], **Benjamin Van Durme**[3]
[1]Johns Hopkins University, [2]Pennsylvania State University, [3]Microsoft Semantic Machines
abel@jhu.edu, chiehyang@psu.edu, sminfo@microsoft.com

Abstract

Collecting data for conversational semantic parsing is a time-consuming and demanding process. In this paper we consider, given an incomplete dataset with only a small amount of data, how to build an AI-powered human-in-the-loop process to enable efficient data collection. A **guided K-best selection** process is proposed, which (*i*) generates a set of possible valid candidates; (*ii*) allows users to quickly traverse the set and filter incorrect parses; and (*iii*) asks users to select the correct parse, with minimal modification when necessary. We investigate how to best support users in efficiently traversing the candidate set and locating the correct parse, in terms of speed and accuracy. In our user study, consisting of five annotators labeling 300 instances each, we find that combining *keyword searching*, where keywords can be used to query relevant candidates, and *keyword suggestion*, where representative keywords are automatically generated, enables fast and accurate annotation.[1]

1 Introduction

Conversational Semantic Parsing (CSP), which aims to turn an utterance into a meaning representation such as an executable program or logical form, plays an important role in task-oriented dialogue systems (Zettlemoyer and Collins, 2009; Cheng et al., 2020; Platanios et al., 2021). In practice, building a complete task-oriented dialogue system requires back-and-forth revision of the meaning representation design. Such a mutable nature makes the data collection process difficult and costly. Depending on the complexity of the meaning representation, annotators might even need further training to equip them with basic domain knowledge about the task. Inspired by Computer Assisted Translation (CAT) (Green et al., 2013, 2014), we would like to know if we can accelerate the annotation process with AI-powered human-in-the-loop interfaces. The main difference between our task and the traditional CAT task lies in the facts that (*i*) only a prototype model trained on a small amount of initial data is available (low-resource setting), leading to limited prediction performance; and (*ii*) annotators have relatively little or no knowledge about the meaning representation.


Natural utterance
When's the lecture scheduled for in May?
Model trained on 1k Dialogues: Accuracy@1 = 0.63
Low-Resource Semantic Parsing Model
Guided K-best Selection
Candidate #1
Candidate #2
⋮
Candidate #100
Canonical utterance
start time of find event called something like "lecture" during May
Interchangeable using an SCFG
Meaning representation
Model trained on 1k Dialogues + K-Best UI: Accuracy@1 = 0.74

```
(Yield :output (:start (singleton
(:results (FindEventWrapperWithDefaults
:constraint (EventDuringRange :event
(Constraint[Event] :subject (?~=
#(String "lecture"))) :range
(FullMonthofMonth :month #(Month
"MAY"))))))))
```

Figure 1: Guided K-best selection approaches achieve accuracy up to 74% when applied to VACSP-1k (Platanios et al., 2021), a conversational semantic parsing model trained on only 1k dialogues. The canonical utterance, used as the annotation target, is interchangeable with the meaning representation using an SCFG.

Because neither the model nor the annotators are 100% accurate in our scenario, we propose the **K-best selection approach** as shown in Figure 1, where we (*i*) generate a set of candidates using a low-resource model; and (*ii*) ask annotators to traverse the set to select the correct parse and only modify it if necessary. This formulation allows annotators to focus on **reading and verification** and thus minimizes the need for annotators to write the complicated meaning representation. While the notion of K-best selection is established (Duan et al., 2016; He et al., 2016), to our knowledge there has been no investigation into optimizing this approach beyond a simple enumeration of candidates ranked by model score. In addition, a standard K-best

[*]Equal contribution. Work performed during an internship at Microsoft Semantic Machines.

[1]Demo video: https://youtu.be/AtbCCYxjKIY

Natural Utterance	Canonical Utterance
Okay, I'll get in touch with them. Can you tell me if Sqirl in Los Angeles has waiter service?	Does "Sqirl in Los Angeles" have waiter service
What do I have on my calendar after 12 pm tomorrow?	find event tomorrow after 12 PM
Can you add a workout with Kim between the sales meeting and dinner?	create event called "workout" starting between find event called something like "sales meeting" to find event called something like "dinner" with recipient "Kim"
The first one. Also make a Stand-up meeting for early next Monday	Yes, create the first one and then create event called "Stand-up meeting" starting next Monday early morning
Yes please do so.	Looks good!
Is it cloudy in Florida?	weather at "Florida" now is cloudy

Table 1: Examples of natural and canonical utterances extracted from the SMCalFlow training set (Semantic Machines et al., 2020). The canonical utterances are generated by the SCFG defined by Shin et al. (2021a)[2].

selection approach may face challenges such as:

- Annotation *speed*: as K grows larger, an annotator needs to spend more time reading the candidate list. Can we organize the candidates list in a way that allows for fast filtering?

- Annotation *accuracy*: early plausible candidates in a ranked list may bias interpretation; an annotator may commit early to a less-than-perfect result without exploring further. Can we encourage exploration without adversely affecting speed?

In this work we demonstrate the validity of these concerns and propose a solution called **guided K-best selection**, consisting of: (*i*) a *search interface* that allows annotators to type keywords and narrow down the K choices, (*ii*) a *keyword suggestion* method that guides the exploration of K-best lists for less experienced users. We show that it is the combination of efficient search and guidance that strikes the optimal balance between accuracy and speed while achieving high annotator satisfaction.

2 Conversational Semantic Parsing

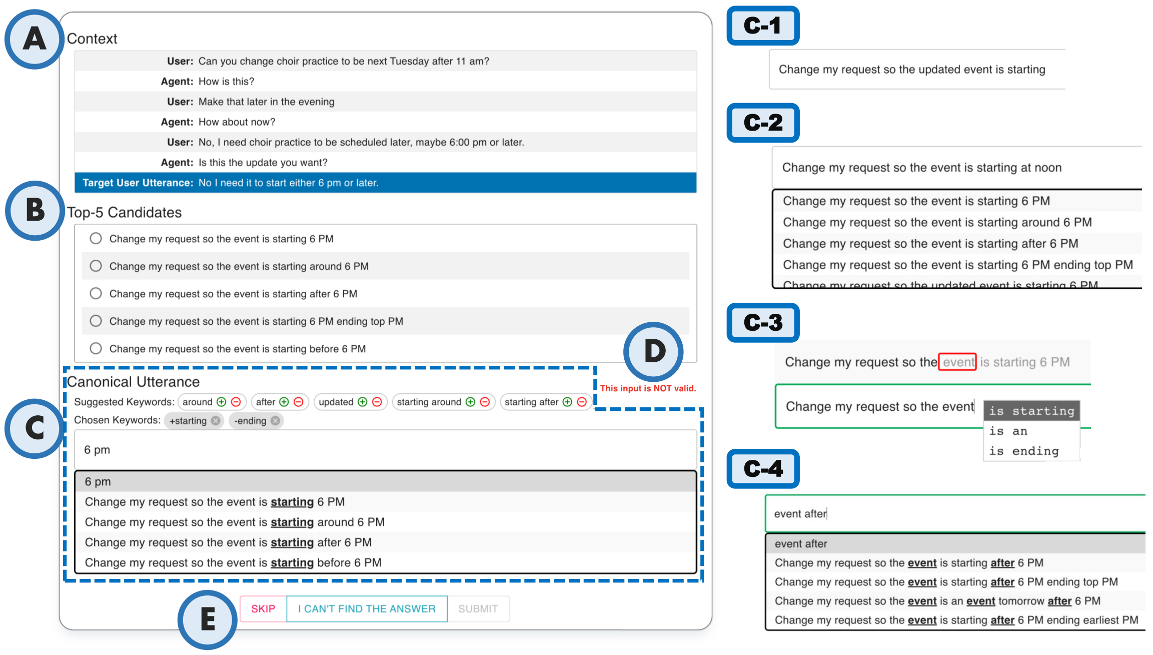
For our study we focus on a version of Conversational Semantic Parsing (Figure 1), where we are given a user's natural utterance, and the goal of the task is to annotate it into a *canonical utterance*. The use of canonical utterances formulates semantic parsing as a paraphrasing task that paraphrases a natural utterance into a "canonical" utterance in a constrained language (Berant and Liang, 2014; Marzoev et al., 2020; Shin et al., 2021a; Wu et al., 2021). A synchronous context-free grammar (*SCFG*) defines a mapping between task-specific meaning representations and their corresponding constrained languages. That is to say, using such an SCFG, a complicated meaning representation can be presented as a human-readable canonical utterance (more similar to natural language) so models can focus on learning how to paraphrase a natural utterance to a canonical utterance. We choose to annotate canonical utterances also because it substantially reduces the task complexity, since annotators no longer need to learn the syntax of the meaning representation itself. We use canonical utterances induced by an SCFG defined in (Shin et al., 2021a). The corresponding meaning representation is defined in the SMCalFlow dataset (Semantic Machines et al., 2020), which contains 41.5K task-oriented dialogues about calendar events, weather, places, and people. Examples of natural and canonical utterances are shown in Table 1.

3 Guided K-best Selection Interfaces

In this section, we describe three proposed and two baseline annotation interfaces. Figure 2 shows their basic components. The user utterance and its context are given in (A) and the K-best candidate list is provided in (C). (C) and its variants provide different functions to help users efficiently get to the correct parse. Note that (A), (B), (D), and (E) are shared across all UIs.

`no-kbest` This interface, shown on Figure 2 (C-1), is an "annotate from scratch" baseline. Users need to type the canonical utterance without seeing the K-best list. They can use (D) to validate whether the current utterance is grammatical.

[2]SCFG implementation is available on the Github repository (Shin et al., 2021b).



Context

User: Can you change choir practice to be next Tuesday after 11 am?
Agent: How is this?
User: Make that later in the evening
Agent: How about now?
User: No, I need choir practice to be scheduled later, maybe 6:00 pm or later.
Agent: Is this the update you want?
Target User Utterance: No I need it to start either 6 pm or later.

Top-5 Candidates

- Change my request so the event is starting 6 PM
- Change my request so the event is starting around 6 PM
- Change my request so the event is starting after 6 PM
- Change my request so the event is starting 6 PM ending top PM
- Change my request so the event is starting before 6 PM

Canonical Utterance

This input is NOT valid.

Suggested Keywords: around ⊕ ⊖ after ⊕ ⊖ updated ⊕ ⊖ starting around ⊕ ⊖ starting after ⊕ ⊖

Chosen Keywords: +starting ⊗ -ending ⊗

6 pm

6 pm
Change my request so the event is **starting** 6 PM
Change my request so the event is **starting** around 6 PM
Change my request so the event is **starting** after 6 PM
Change my request so the event is **starting** before 6 PM

SKIP | I CAN'T FIND THE ANSWER | SUBMIT

Change my request so the updated event is starting

Change my request so the event is starting at noon

Change my request so the event is starting 6 PM
Change my request so the event is starting around 6 PM
Change my request so the event is starting after 6 PM
Change my request so the event is starting 6 PM ending top PM
Change my request so the updated event is starting 6 PM

Change my request so the event is starting 6 PM

Change my request so the event

```
is starting
is an
is ending
```

event after

event after
Change my request so the **event** is starting **after** 6 PM
Change my request so the **event** is starting **after** 6 PM ending top PM
Change my request so the **event** is an **event** tomorrow **after** 6 PM
Change my request so the **event** is starting **after** 6 PM ending earliest PM

Figure 2: The main components of `search-keywords` and other interfaces. (A), (B), (D) and (E) are shared across all interfaces. (A) shows the dialog context and the target user utterance to annotate; (B) shows the top 5 candidates to serve as the options and hints; (D) indicates whether the current input is grammatical, i.e. can be parsed with an SCFG (the latency of the grammar verification function is around 70 ms, which is quick enough for real-time querying); (E) presents buttons for submitting the current task, "skipping" it to work on later, or "escalating" it for manual annotation (via the "I can't find the answer" button, which declares that the correct parse is not in the top K). `Search-keywords` (C) suggests a set of keywords for users to query relevant candidates on. `No-kbest` (C-1) only provides an input box for manually entering the annotation. `Scroll` (C-2) simply presents all the candidates for users to select. `Autocomplete` (C-3) shows both a full sentence completion and possible next chunks of the tokens. `Search` (C-4) allows users to enter keywords to query relevant candidates.

`scroll` Figure 2 (C-2) shows the `scroll` interface. `scroll` serves as another baseline in our experiments. We simply present all the candidates ordered by their model scores. Users are able to use their mouse or keyboard to traverse the list.

`autocomplete` As shown in Figure 2 (C-3), `autocomplete` shows a full sentence completion above the input area and possible next chunks of tokens next to the cursor. To generate the suggestions, we insert all candidates into a trie (Browning, 2021). The full sentence completion is the one that satisfies the prefix constraint and has the highest model score; and the next chunks of tokens are generated by traversing the trie until different tokens appear. The red box serves as a cursor around the current token in the full sentence completion. When using `autocomplete`, users essentially explore the K candidates by traversing the trie.

`search` Figure 2 (C-4) shows the `search` interface. It aims to break the *left-to-right* nature of `autocomplete`, where users need to traverse the trie in a certain order. `search` allows users to enter keywords in arbitrary order to remove irrelevant candidates. After entering the keywords, valid candidates ordered by the model scores will be shown below the input area. The matched keywords are highlighted in bold and underlined for quick reference. We use flexsearch (Wilkerling, 2021) to index the candidate list in the frontend UI to further reduce the latency.

`search-keywords` As shown in Figure 2 (C), `search-keywords` extends the `search` interface by showing a list of top 5 discriminative keywords. These keywords are used to narrow down the current candidates. They also give users a rough overview of the candidate list and the annotation grammar. Users can choose to include (+) or exclude (−) the keyword in the correct parse.

To provide suggestions, we develop a Keyword Suggestion (KS) method inspired by post-decoding clustering (PDC) from (Ippolito et al., 2019). We similarly perform k-means clustering over the K-

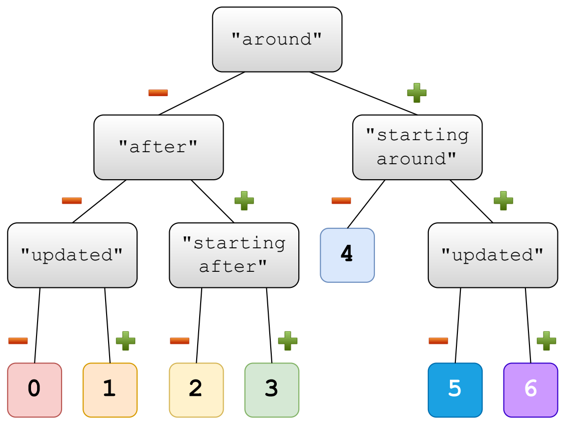


Figure 3: KS example. We generate an explanation tree over a k-means clustering (k=7) and use $k' = 5$ unique intermediate nodes' n-grams as suggested keywords.

	Top-5	**Top-20**	**Top-100**	**Escalate**
Stratified	25%	25%	25%	25%
True	76%	6%	4%	14%

Table 2: We apply stratified sampling to control the distribution of gold answers. **Escalate** is the case where the gold parse is not in top 100. The **True** distribution gives each stratum's real distribution in the dev set.

best list and choose the candidate with the highest model score to represent each cluster. This distills the original K candidates into fewer but more diverse candidates, where $k \ll K$. In addition, we employ a cluster explanation technique recently proposed by Dasgupta et al. (2020) to further distill the k diverse candidates into k' *keywords*. This is done by approximating the k clusters' decision boundaries (which are arbitrarily shaped) with k axis-aligned rectangles. As a result, this approximated k-means clustering can be summarized with a binary tree, consisting of k leaf nodes and at most $k - 1$ intermediate split nodes. Split nodes correspond to n-grams ($n = 1, 2, 3$) formed from the canonical representations of candidate parses. We use the set of all unique split nodes' n-grams to form a set of suggested keywords. Thus, the k diverse candidates are distilled even further into k' keywords, $k' < k \ll K$. An example of this process is given on Figure 3. Finally, the k' keywords are re-ranked based on their *discriminativeness*, or how evenly they split the current candidates, and the 5 most discriminative keywords are shown in the interface. This allows combining keyword suggestion with the `search` interface. In Section 4.3, we compare keyword suggestion with other mechanisms to guide annotators.

4 Experiments

4.1 Protocol

Data 300 utterances were sampled from the SMCalFlow development set (Semantic Machines et al., 2020). For each utterance, we used the state-of-the-art conversational semantic parser VACSP (Platanios et al., 2021) to generate $K = 100$ candidate parses.[3] To simulate a low-resource setting, we used the variant of VACSP trained on 1k dialogues (VACSP-1k). We sampled utterances according to the stratified distribution from Table 2 in order to represent multiple rank settings equally. For **Escalate** samples, where the gold answer was not presented in the candidate list, we expected the participants to either choose the "I can't find the answer" option,[4] or edit one of the candidates to obtain the correct parse.

Participants A total of 5 participants joined the experiment. All participants were not previously exposed to the canonical language and the proposed interfaces. To help annotators get familiar with the canonical grammar, they were asked to read 300 (user utterance, canonical utterance) pairs.[5] Participants were then randomly assigned to a particular interface and data split. Each interface was used along with a different data split to reduce potential bias. After finishing, participants filled out a questionnaire, evaluating (*i*) interface preference on a 5-point Likert scale, (*ii*) cognitive load using the NASA Task Load Index (Hart, 2006) on a 7-point Likert scale, and (*iii*) free-text suggestions.

4.2 Interface Comparison

We compare interfaces from (*i*) the requester's perspective by evaluating annotation accuracy and time and (*ii*) the annotator's perspective by evaluating their UI preference across multiple criteria.

Accuracy and time Table 3 shows the exact match accuracy and the time usage per utterance.

To verify how accuracy and time differ depending on the difficulty of each instance, we measure the rank of the gold parse in the K-best list, and aggregate the results over three non-overlapping set-

[3]We used Accuracy@K as a proxy to decide the best K. We chose $K = 100$ since larger values of K did not substantially improve Accuracy@K with our prototype model: e.g. Accuracy@200 was only 0.8% higher than Accuracy@100.

[4]Practically, **Escalate** means sending this hard instance to experienced and knowledgeable annotators to handle.

[5]The 300 pairs were selected to represent a diverse set of functions in the canonical and meaning representations.

	Exact Match Accuracy ↑							Median Time (sec) ↓				
	Top-5	Top-20	Top-100	Escalate	Escalate$_m$	All	True	Top-5	Top-20	Top-100	Escalate	All
No-KBest	.411	.189	.123	.400	.067	.197	.339	56.13	73.17	97.48	74.29	69.43
Scroll	<u>.880</u>	.320	<u>.213</u>	<u>.453</u>	.067	.370	.706	13.00	25.84	<u>26.47</u>	<u>30.23</u>	24.73
Autocomplete	**.919**	<u>.370</u>	**.333**	.427	.067	**.422**	**.743**	13.71	26.01	30.02	31.47	25.53
Search	.878	.320	<u>.213</u>	.400	<u>.080</u>	.373	.707	**8.48**	**19.09**	**17.16**	**19.55**	**16.02**
Search-Keywords	<u>.880</u>	**.419**	<u>.213</u>	**.480**	**.093**	<u>.401</u>	<u>.716</u>	<u>12.78</u>	<u>24.51</u>	36.26	31.15	<u>23.91</u>

Table 3: Accuracy and time usage of the baseline and proposed interfaces. In the last stratum, Escalate, in which answer is not provided, we present two values: **Escalate$_m$**, where only matching the gold answer is correct, and **Escalate**, where in addition to that selecting "I Can't Find The Answer" is also treated as correct. **All** is the mean accuracy over all the strata. **True** stands for the true accuracy weighted by the true distribution in the dev set (Table 2). Note **All** and **True** are computed using **Escalate$_m$**. **Bolded** is the best result; <u>underlined</u> is the second-best result. `Autocomplete` achieves the highest accuracy and `Search` help reduce time usage up to 35% compared to `Scroll`. `Search-keywords` strikes the balance between accuracy and time usage.

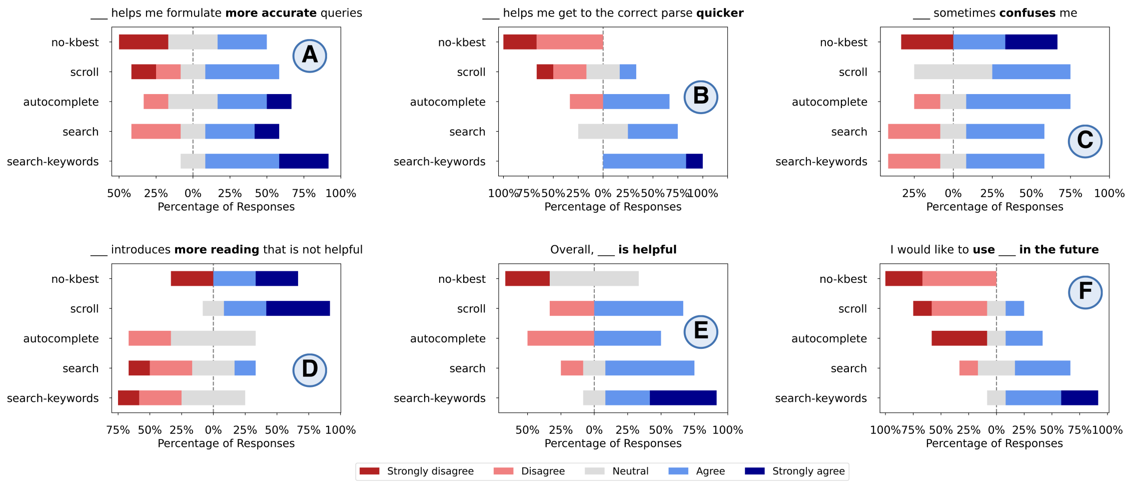


Figure 4: Summary of the user feedback on the interface preference. Across six evaluation criteria, the users preferred the proposed UIs over baselines, with `search-keywords` being their top choice across all criteria.

tings: "Top-5", "Top-20", and "Top-100",[6] which correspond to varying difficulties of instances.

We further compute the *true accuracy* by estimating the real distribution of the strata in our dataset to compare with Platanios et al. (2021)'s VACSP-1k results. As shown in the **True** accuracy column in Table 3, by introducing humans into the semantic parsing process, no matter which K-best interface they use, the performance always improves (the **True** accuracy of the VACSP-1k model is 0.630). When comparing each interface, `autocomplete` achieves the highest accuracy overall but it also takes a longer time. We hypothesize that the accuracy gain comes from the fact that annotators are required to review tokens individually. Compared to `scroll`, `search` does not substantially improve the accuracy, but it does reduce the overall time usage by up to 35%. `No-KBest` shows that, with only the grammar verification, annotators are much slower (69s vs 25s) and less accurate (.339 vs .706) even compared to `scroll` (see Appendix B for more analysis on annotation time distribution). Overall, `search-keywords` strikes a balance between the trade-offs, being generally either best or close to best in both accuracy and time usage.

User feedback Figure 4 summarizes user answers to UI preference questions from the questionnaire.[7] Annotators generally preferred the proposed interfaces to the `no-kbest` and `scroll` baselines: they found that the proposed UIs enable them to be more accurate (A) and faster (B), as well as requiring less unnecessary reading (D). The level of confusion was roughly the same across UIs (C), perhaps due to non-UI related factors (such as an-

[6] "Top-5" corresponds to the rank in $[1, 5]$ "Top-20" corresponds to $[6, 20]$, and "Top-100" corresponds to $[21, 100]$.

[7] More feedback, incl. excerpts of free-form suggestions and NASA task load index results, is provided in Appendix C.

	Oracle simulation results ($k = 5$) Average number of turns ↓				**Human annotation results** ($k = 5$) Median time (sec) ↓			
	Top-5	**Top-20**	**Top-100**	**All**	**Top-5**	**Top-20**	**Top-100**	**All**
KS (ours)	1.10	**2.39**	**2.80**	**1.24**	**15.30**	46.99	**48.20**	**36.71**
PDC (k-means, canonical)	1.11	2.40	2.84	**1.24**	25.09	73.23	55.42	52.94
PDC (agglomerative, canonical)	1.16	2.73	3.10	1.31	—	—	—	—
PDC (agglomerative, meaning)	1.15	2.68	2.91	1.29	—	—	—	—
Scroll	**1.00**	2.63	7.75	1.33	24.18	**42.30**	56.37	37.21

Table 4: Comparison of K-best guidance strategies: by suggesting keywords (top row), diverse candidates (middle rows), or all K candidates (bottom row). In both oracle and human settings, $k = 5$ candidates are displayed per each interaction turn. **Bolded** is the best result, underlined is the second-best result. While KS and PDC perform similarly in the oracle setting, the former leads to faster annotation when tested with real human annotators.

notation grammar or stratified distribution of examples). The annotators also preferred the proposed UIs for future use (F), with `search-keywords` being the most preferred one by a large margin.

4.3 Guidance Comparison

We compare our keyword suggestion (KS) method, based on explainable k-means clustering, with the PDC algorithm from (Ippolito et al., 2019) and the `scroll` baseline from Section 3. We compare multiple variants of PDC, exploring whether agglomerative clustering (based on string edit distance) or k-means is better, and whether canonical or meaning representation is better. All algorithms are used interactively: e.g., during one turn of PDC, a user would pick one out of k clusters (represented by the top scoring parse each) that looks most correct to them, and on the next turn, they would only see candidates from the cluster chosen previously. In KS, one turn corresponds to choosing whether a single suggested keyword (e.g. "create event") should or should not be included in the correct parse while seeing the currently best scoring k candidates. Finally, in `scroll` the user simply scrolls over K candidates using a size-k window. To achieve a fair comparison and manageable workload, all methods display $k = 5$ candidates per interaction turn.

We evaluate guidance methods in two ways: with a simulated (oracle) user, and with a pool of human annotators described in Section 4.1. The simulated user will always make the best choice at each interaction turn: that is, pick the keyword or the candidate parse that will be included in, or will be closest to, the best parse. Human annotators, however, might make mistakes. Thus, in all interfaces, they are allowed to go several turns back and fix mistakes before submitting. We report the number of turns for the simulated user and the wall annotation time for human annotators in Table 4.

Oracle simulation results Table 4 (left) shows several trends: first, adding explanations and "coarsening" clusters' decision boundary in KS does not hurt annotation speed, compared to the otherwise similar "PDC (k-means, canonical)". Second, both KS and PDC substantially decrease the gap between the easiest Top-5 and the hardest Top-100 settings observed for `scroll`, contributing to a more predictable annotator experience. Finally, comparing the results across PDC variants, we can conclude that k-means over canonical representation is a reasonable default setting, and we lock on to it for human annotation experiments.

Human annotation results Table 4 (right) agrees with the simulation finding that KS and PDC help decrease the gap between Top-5 and Top-100 settings, albeit to a lesser extent than in the simulation experiment. Most notable is the difference in speed between KS and PDC: while in simulation it was only marginal, here the lack of explanations substantially slows the human annotators down. In addition, in the post-experiment survey, the annotators were confused by "non-intuitive similarity relations" and "too much extra reading" of the clustering-based PDC while praising the keyword-based KS for being "intuitive" and "engaging".

5 Related Work

5.1 Interactive Semantic Parsing

K-best selection is aligned with *interactive semantic parsing*. These approaches assume access to an existing parsing model, and to a user that provides corrections in binary (Clarke et al., 2010; Artzi and Zettlemoyer, 2013; Iyer et al., 2017), multiple-choice (Iyer et al., 2017; Gur et al., 2018; Yao et al., 2019), or natural language form (Elgohary et al., 2021). Many of these methods also rely on separate trainable modules or parsing model's parameters to identify inference steps the parser is most uncertain about. In contrast, K-best selection is parameter-free (making it well-suited for low-resource settings).

5.2 Computer-Assisted Translation

Our search and autocomplete UIs are motivated by *computer-assisted translation*. There are three main directions in CAT. **Post-Editing (PE)** asks users to revise and verify a machine translated text (Green et al., 2013; Aranberri et al., 2014; Toral et al., 2018; Herbig et al., 2020; Lee et al., 2021). **Interactive Translation Prediction (ITP)** suggests translations dynamically based on users' input (Langlais et al., 2000; Foster et al., 2002; Bender et al., 2005; Barrachina et al., 2009; Koehn, 2009; Alabau et al., 2014; Green et al., 2014; Cheng et al., 2016). Both PE and ITP assume users' translations are nearly perfect which is not always the case in CSP. **Iterative translation (IT)** asks two groups of monolingual speakers to iterate over the translation back and forth to improve translation quality (Morita and Ishida, 2009; Hu et al., 2010, 2011). Although IT ensures quality, its low throughput (2.5 to 6 times slower compared to professional translators (Hu et al., 2011)) prevents us from using it.

5.3 Diverse Generation

Our keyword suggestion mechanism is motivated by *diverse text generation*. Typical strategies for improving the collective diversity (Hu et al., 2019) of the output candidates include: modifications to beam search (Vijayakumar et al., 2018; Tam, 2020), modifications to the sampling method (Fan et al., 2018; Holtzman et al., 2020), stratified sampling based on semantic codes (Weir et al., 2020), and post-decoding clustering (Kriz et al., 2019; Ippolito et al., 2019). The latter approach involves over-generating candidates by using a large beam size, clustering the final candidates, and selecting one or a few representative candidates per cluster.

6 Conclusion

In this paper, we tackled the challenge of efficient data collection for conversational semantic parsing. In the presence of little available training data, we propose human-in-the-loop interfaces for **guided K-best selection**, using a prototype model trained on limited data. Guided K-best selection interfaces generate a set of possible candidates with functions for fast traversal and ask annotators to select the correct parse. User studies show that combining keyword search functionality with a keyword suggestion system strikes an optimal balance between annotation accuracy and speed.

References

Vicent Alabau, Christian Buck, Michael Carl, Francisco Casacuberta, Mercedes García-Martínez, Ulrich Germann, Jesús González-Rubio, Robin Hill, Philipp Koehn, Luis A Leiva, et al. 2014. Casmacat: A computer-assisted translation workbench. In *Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 25–28.

Nora Aranberri, Gorka Labaka, Arantza Diaz de Ilarraza, and Kepa Sarasola. 2014. Comparison of post-editing productivity between professional translators and lay users. In *Proceedings of the 11th Conference of the Association for Machine Translation in the Americas*, pages 20–33.

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62.

Sergio Barrachina, Oliver Bender, Francisco Casacuberta, Jorge Civera, Elsa Cubel, Shahram Khadivi, Antonio Lagarda, Hermann Ney, Jesús Tomás, Enrique Vidal, and Juan-Miguel Vilar. 2009. Statistical approaches to computer-assisted translation. *Computational Linguistics*, 35(1):3–28.

Oliver Bender, Saša Hasan, David Vilar, Richard Zens, and Hermann Ney. 2005. Comparison of generation strategies for interactive machine translation. In *Proceedings of the 10th EAMT Conference: Practical applications of machine translation*, Budapest, Hungary. European Association for Machine Translation.

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1415–1425.

Lyndsey Browning. 2021. Github repository: trie-prefix-tree.

Mia Xu Chen, Benjamin N. Lee, Gagan Bansal, Yuan Cao, Shuyuan Zhang, Justin Lu, Jackie Tsay, Yinan Wang, Andrew M. Dai, Zhifeng Chen, Timothy Sohn, and Yonghui Wu. 2019. Gmail smart compose: Real-time assisted writing. In *Proceedings* *of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2287–2295, New York, NY, USA. Association for Computing Machinery.

Jianpeng Cheng, Devang Agrawal, Héctor Martínez Alonso, Shruti Bhargava, Joris Driesen, Federico Flego, Dain Kaplan, Dimitri Kartsaklis, Lin Li, Dhivya Piraviperumal, Jason D. Williams, Hong Yu, Diarmuid Ó Séaghdha, and Anders Johannsen. 2020. Conversational semantic parsing for dialog state tracking. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8107–8117, Online. Association for Computational Linguistics.

Shanbo Cheng, Shujian Huang, Huadong Chen, Xinyu Dai, and Jiajun Chen. 2016. Primt: A pick-revise framework for interactive machine translation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1240–1249.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the world's response. In *Proceedings of the fourteenth conference on computational natural language learning*, pages 18–27.

Sanjoy Dasgupta, Nave Frost, Michal Moshkovitz, and Cyrus Rashtchian. 2020. Explainable k-means and k-medians clustering. In *Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria*, pages 12–18.

Manjuan Duan, Ethan Hill, and Michael White. 2016. Generating disambiguating paraphrases for structurally ambiguous sentences. In *Proceedings of the 10th Linguistic Annotation Workshop held in conjunction with ACL 2016 (LAW-X 2016)*, pages 160–170.

Ahmed Elgohary, Christopher Meek, Matthew Richardson, Adam Fourney, Gonzalo Ramos, and Ahmed Hassan Awadallah. 2021. NL-EDIT: Correcting semantic parse errors through natural language interaction. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5599–5610, Online. Association for Computational Linguistics.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898, Melbourne, Australia. Association for Computational Linguistics.

George Foster, Philippe Langlais, and Guy Lapalme. 2002. User-friendly text prediction for translators. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 148–155. Association for Computational Linguistics.

Spence Green, Jason Chuang, Jeffrey Heer, and Christopher D Manning. 2014. Predictive translation memory: A mixed-initiative system for human language translation. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 177–187.

Spence Green, Jeffrey Heer, and Christopher D Manning. 2013. The efficacy of human post-editing for language translation. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 439–448.

Izzeddin Gur, Semih Yavuz, Yu Su, and Xifeng Yan. 2018. Dialsql: Dialogue based structured query generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1339–1349.

Sandra G. Hart. 2006. Nasa-task load index (nasa-tlx); 20 years later. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 50(9):904–908.

Luheng He, Julian Michael, Mike Lewis, and Luke Zettlemoyer. 2016. Human-in-the-loop parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2337–2342.

Nico Herbig, Tim Düwel, Santanu Pal, Kalliopi Meladaki, Mahsa Monshizadeh, Antonio Krüger, and Josef van Genabith. 2020. Mmpe: A multi-modal interface for post-editing machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1691–1702.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Chang Hu, Benjamin B Bederson, and Philip Resnik. 2010. Translation by iterative collaboration between monolingual users. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, pages 54–55.

Chang Hu, Benjamin B Bederson, Philip Resnik, and Yakov Kronrod. 2011. Monotrans2: A new human computation system to support monolingual translation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1133–1136.

J Edward Hu, Abhinav Singh, Nils Holzenberger, Matt Post, and Benjamin Van Durme. 2019. Large-scale, diverse, paraphrastic bitexts via sampling and clustering. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 44–54.

Daphne Ippolito, Reno Kriz, João Sedoc, Maria Kustikova, and Chris Callison-Burch. 2019. Comparison of diverse decoding methods from conditional language models. In *Proceedings of the 57th*

Annual Meeting of the Association for Computational Linguistics, pages 3752–3762, Florence, Italy. Association for Computational Linguistics.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973, Vancouver, Canada. Association for Computational Linguistics.

Rebecca Knowles, Marina Sanchez-Torron, and Philipp Koehn. 2019. A user study of neural interactive translation prediction. *Machine Translation*, 33(1):135–154.

Philipp Koehn. 2009. A process study of computer-aided translation. *Machine Translation*, 23(4):241–263.

Reno Kriz, João Sedoc, Marianna Apidianaki, Carolina Zheng, Gaurav Kumar, Eleni Miltsakaki, and Chris Callison-Burch. 2019. Complexity-weighted loss and diverse reranking for sentence simplification. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3137–3147, Minneapolis, Minnesota. Association for Computational Linguistics.

Philippe Langlais, George Foster, and Guy Lapalme. 2000. TransType: a computer-aided translation typing system. In *ANLP-NAACL 2000 Workshop: Embedded Machine Translation Systems*.

Dongjun Lee, Junhyeong Ahn, Heesoo Park, and Jaemin Jo. 2021. IntelliCAT: Intelligent machine translation post-editing with quality estimation and translation suggestion. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 11–19, Online. Association for Computational Linguistics.

Alana Marzoev, Samuel Madden, M Frans Kaashoek, Michael Cafarella, and Jacob Andreas. 2020. Unnatural language processing: Bridging the gap between synthetic and natural language data. *arXiv preprint arXiv:2004.13645*.

Daisuke Morita and Toru Ishida. 2009. Designing protocols for collaborative translation. In *Proceedings of the 12th International Conference on Principles of Practice in Multi-Agent Systems*, PRIMA '09, page 17–32, Berlin, Heidelberg. Springer-Verlag.

Emmanouil Antonios Platanios, Adam Pauls, Subhro Roy, Yuchen Zhang, Alexander Kyte, Alan Guo, Sam Thomson, Jayant Krishnamurthy, Jason Wolfe, Jacob Andreas, and Dan Klein. 2021. Value-agnostic conversational semantic parsing. In *ACL-IJCNLP 2021*.

Semantic Machines, Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, Hao Fang, Alan Guo, David Hall, Kristin Hayes, Kellie Hill, Diana Ho, Wendy Iwaszuk, Smriti Jha, Dan Klein, Jayant Krishnamurthy, Theo Lanman, Percy Liang, Christopher H. Lin, Ilya Lintsbakh, Andy McGovern, Aleksandr Nisnevich, Adam Pauls, Dmitrij Petters, Brent Read, Dan Roth, Subhro Roy, Jesse Rusak, Beth Short, Div Slomin, Ben Snyder, Stephon Striplin, Yu Su, Zachary Tellman, Sam Thomson, Andrei Vorobev, Izabela Witoszko, Jason Wolfe, Abby Wray, Yuchen Zhang, and Alexander Zotov. 2020. Task-oriented dialogue as dataflow synthesis. *Transactions of the Association for Computational Linguistics*, 8:556–571.

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

Richard Shin, Christopher H. Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021b. Github repository: Constrained language models yield few-shot semantic parsers.

Yik-Cheung Tam. 2020. Cluster-based beam search for pointer-generator chatbot grounded by knowledge. *Computer Speech & Language*, 64:101094.

Antonio Toral, Martijn Wieling, and Andy Way. 2018. Post-editing effort of a novel with statistical and neural machine translation. *Frontiers in Digital Humanities*, 5:9.

Ashwin K Vijayakumar, Michael Cogswell, Ramprasath R Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. 2018. Diverse beam search: Decoding diverse solutions from neural sequence models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7371–7379. AAAI.

Nathaniel Weir, João Sedoc, and Benjamin Van Durme. 2020. COD3S: Diverse generation with discrete semantic signatures. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5199–5211, Online. Association for Computational Linguistics.

Thomas Wilkerling. 2021. Github repository: flexsearch.

Shan Wu, Bo Chen, Chunlei Xin, Xianpei Han, Le Sun, Weipeng Zhang, Jiansong Chen, Fan Yang, and Xunliang Cai. 2021. From paraphrasing to semantic parsing: Unsupervised semantic parsing via synchronous semantic decoding. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint*

Conference on Natural Language Processing (Volume 1: Long Papers), pages 5110–5121, Online. Association for Computational Linguistics.

Ziyu Yao, Yu Su, Huan Sun, and Wen-tau Yih. 2019. Model-based interactive semantic parsing: A unified framework and a text-to-SQL case study. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5447–5458, Hong Kong, China. Association for Computational Linguistics.

Luke Zettlemoyer and Michael Collins. 2009. Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 976–984, Suntec, Singapore. Association for Computational Linguistics.

A Discussion

In this section, we share additional lessons we learned while interacting with expert annotators in the preliminary experiments and adapting our final proposed interfaces according to their feedback.

Latency is a critical issue when generating completions dynamically. We initially experimented with the autocomplete interface suggesting completions fully **dynamically**, similar to recent works on CAT (Green et al., 2014; Knowles et al., 2019). To achieve this, we periodically sent users' inputs to the backend BART model (Shin et al., 2021a) that would perform completions using beam search decoding. We found the latency of the BART model to be around 300-600 ms even with the beam size of one and constrained decoding turned off (turning the constrained decoding on ensures the generated completions are grammatical but doubles the latency). In a small preliminary study, participants generally noted the dynamic interface to be "laggy". Similar issues also happened in prior studies, e.g. Green et al. (2014) noticed that users deemed the interface as "sluggish" unless the latency was reduced to less than 300 ms by using the phrase-based decoding algorithm to reduce the search space; Chen et al. (2019) examined the latency of LSTM as well as Transformer and concluded that Transformer's high latency was not suitable for production despite the performance gain. We thus concluded that generating completions dynamically is infeasible and directed our study towards K-best selection.

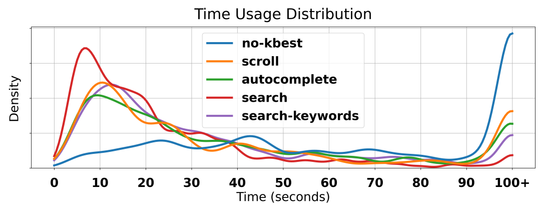


Figure 5: KDE plot of the time usage for the baseline and proposed interfaces. `Search` has a higher distribution between 5-20 seconds; The long tail is merged and forms another peak in 100+ where we can clearly see that `no-kbest` $\gg$ `scroll` $>$ `autocomplete` $>$ `search-keywords` $>$ `search`.

Users are not able to produce a full canonical utterance from scratch. Compared to CAT tasks (Section 5.2), one of the difficulties of annotating semantic representations is that the canonical language is hard and subject to change. In CAT, people assume that the produced translation is always valid which is not always true for semantic representation annotation tasks. Annotators found it substantially harder to author a complete annotation from scratch using the `no-kbest` interface (Figure 4), as compared to selecting from a list of K options with all other UIs. We thus believe that a K-best framework should be a preferred approach as it relies on people's ability to **read and verify** the candidates rather than producing an answer from scratch.

B Additional Experiments

Oracle simulation results In addition to $k = 5$ (Section 4.3), we experiment with displaying less ($k = 2$) and more ($k = 10$) items per turn. The results are shown on Table 5. When displaying $k = 2$ (left) items per turn, the difference between Top-100 "tail" performance of the scroll baseline vs proposed methods is very high, thus making proposed methods more predictable on the tail. For $k = 10$ (right) items per turn, this difference is more leveled and the overall performance is essentially the same across all methods, at the expense of increased reading per turn. In the human experiments, we chose $k = 5$ to strike balance between workload per turn and reasonable number of turns.

Overall time usage distribution In addition to the user study results presented in Section 4.2, we show the kernel density estimation (KDE) plot of the time usage distribution in Figure 5. The KDE

	Oracle simulation results ($k = 2$) Average number of turns ↓				**Oracle simulation results** ($k = 10$) Average number of turns ↓			
	Top-5	**Top-20**	**Top-100**	**All**	**Top-5**	**Top-20**	**Top-100**	**All**
KS (ours)	<u>1.19</u>	<u>4.22</u>	<u>5.90</u>	<u>1.53</u>	<u>1.06</u>	<u>1.79</u>	<u>2.09</u>	**1.14**
PDC (k-means, canonical)	<u>1.19</u>	**4.19**	**5.85**	**1.52**	<u>1.06</u>	1.81	2.11	**1.14**
PDC (agglomerative, canonical)	1.24	5.38	7.14	1.68	1.10	1.92	2.10	<u>1.18</u>
PDC (agglomerative, meaning)	1.23	5.00	6.55	1.63	1.10	1.94	**2.03**	<u>1.18</u>
Scroll	**1.09**	5.65	18.58	1.96	**1.00**	**1.45**	4.15	**1.14**

Table 5: Additional oracle simulation results with $k = 2$ (left) and $k = 10$ (right) candidates displayed per turn. **Bolded** is the best result, <u>underlined</u> is the second-best result.

plots are produced using seaborn[8] with bandwidth adjustment `bw_adjust` = 3 and `clip` = (0, 100). Note that we clip the time to the range $[0, 100]$ to better display the distribution tail. We find that a huge portion of `search` locates within 5–20 seconds showing that users indeed can finish the task much faster. In another peak (100+ seconds), the distribution clearly shows `no-kbest` $\gg$ `scroll` $>$ `autocomplete` $>$ `search-keywords` $>$ `search` meaning that `no-kbest` takes much longer time in general; and a higher portion from `scroll` and `autocomplete` takes much longer time to finish; whereas `search-keywords` and `search` have fewer such cases. This peak is contributed almost evenly by the four different strata, perhaps because users tend to read through all candidates to make sure they get the right answer.

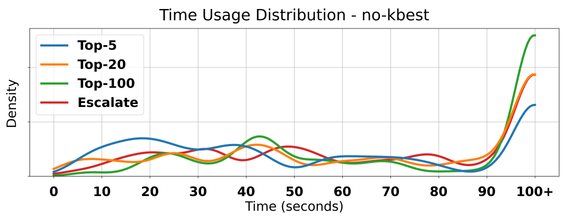
Time usage distribution per interface We plot the time usage distribution using KDE for the proposed interfaces to illustrate the time usage for each stratum. Again, the KDE plots are produced using seaborn with bandwidth adjustment `bw_adjust` = 3 and `clip` = (0, 100). The plots shown on Figure 6 tell us that, for K-best interfaces, only Top-5 shows a different behavior where tasks can be mostly finished within 10 seconds; while Top-20, Top-100, and Escalate have very similar distributions. We hypothesize this is because we explicitly show the top 5 candidates in the interface (Figure 2 (B)). When the gold parse is not in the top 5 candidate list, annotators go through a similar process to find the answer, resulting in a similar time usage distribution for Top-20, Top-100, and Escalate. The `no-kbest` shows that without any supports, a huge portion of the tasks took more than 100 seconds to finish.

[8] `https://seaborn.pydata.org`

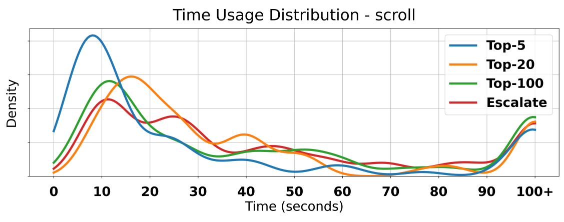
C Additional User Feedback

NASA Task Load Index Figure 7 summarizes user responses to a NASA Task Load Index questionnaire (Hart, 2006) that evaluates participants' subjective workload across six dimensions. The original 7-point Likert scale was mapped to a 5-point scale for conciseness: "very low", "very high", and "medium" labels were preserved, and the four intermediate labels were mapped into two. This feedback was not collected for `search-keywords`; for `autocomplete` and `search` we observe that, compared to baselines, users report lower temporal demand and higher performance, which correspond to higher perceived speed and accuracy, respectively. This agrees with the interface preference feedback (Figure 4) and quantitative results of our user study (Table 3).

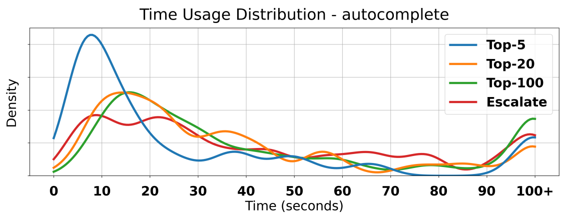
Free-form suggestions We collected participants' free-form suggestions by asking four questions: (*i*) "I like the provided function because ...", (*ii*) "I do not like the provided function because ...", (*iii*) "I think provided function can be improved by ...", and (*iv*) "I would like to have some other functions such as ...". Tables 6 to 9 summarize the responses. In general, participants expressed positive impression towards `autocomplete`, `search`, and `search-keywords`. Users especially like `search-keywords` as it helps quickly narrow down the options; gives insights into grammar; and even when the suggestions fail, it can be figured out very quickly and do not cause a huge negative impact (Table 6). Participants suggested to add more keywords and allow pulling keywords from natural utterances to further improve `search-keywords` (Table 8). We also found that participants believe a grammar guide would improve their annotation process (Table 9) which might be infeasible in a rapid prototyping setup.



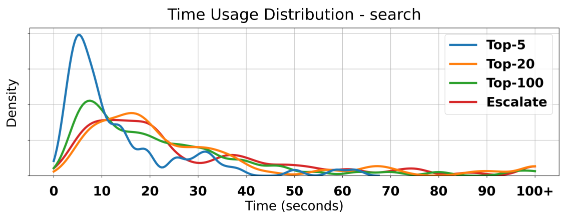
(a) Time usage for `no-kbest`.



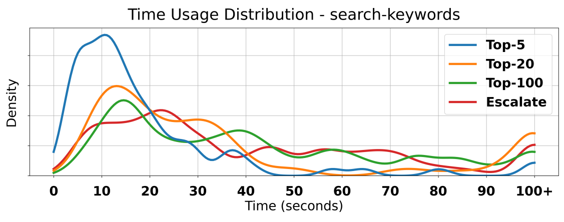
(b) Time usage for `scroll`.



(c) Time usage for `autocomplete`.



(d) Time usage for `search`.



(e) Time usage for `search-keywords`.

Figure 6: For interfaces with K-best supports ((a), (b), (c), (d)), only Top-5 has a different shape of distribution where there is a much higher peak around 10 seconds; Top-20, Top-100, and Escalate have very similar time distribution which suggests that annotators might need to go through the same searching process no matter which stratum it is. The distribution of `no-kbest` is relatively **flat** with a huge peak in 100+, meaning that it takes much more time to finish in general.

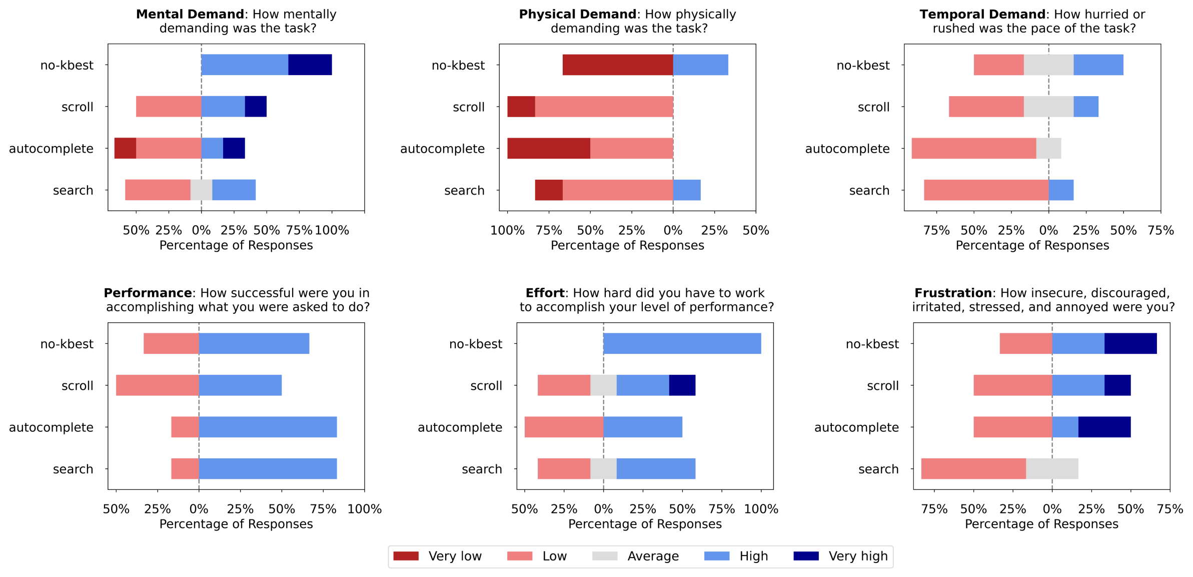


Figure 7: Summary of the user feedback on NASA Task Load Index (Hart, 2006) across six different criteria.

I **like** the provided function because ...	
`scroll`	"You can **easily comb** through the many **variations of grammar**."
`autocomplete`	"... the ability to create the correct grammar with assistance was very helpful and **lowered the frustration** overall" "... it has enough context/information to **quickly determine the right path forward**." "It seemed to be focused and direct ... It seemed to give the **right amount of predictive assistance**."
`search`	"Being able to **narrow down the options** based on a **keyword** is nice, especially in cases **where the bot mostly gets it wrong right at the start**" "... nicely and succinctly. It seems more **intuitive** to use ..."
`search-keywords`	"... the suggested keywords can **very quickly narrow down the list of displayed options** (without having to type in a single search parameter myself) so that **the correct one is easy to locate**." "It was useful for **narrowing things down quicker** and helped **minimize typos** in the search." "It **gave insight into the grammar**, but **was also very efficient**. It allowed myself **more control over the options** I was seeing, meaning it was much more streamlined and was very fast." "Even in those cases where the suggested keywords don't end up helping much, **it only takes a few second to figure that out**. It has minimal impact on the interface, too, so its presence doesn't hurt even when it doesn't help."

Table 6: Free-form suggestions for question "I like the provided function because ...".

I **do not like** the provided function because ...	
`scroll`	"the scrollable list is just **annoying to look at**." "It is **hard to navigate**. And makes little sense."
`autocomplete`	"... sometimes it presenting two options that **both look like they could be valid side-by-side** ..."
`search`	"... it sometimes **highlights in odd ways that decreases readability**."
`search-keywords`	"They weren't usually presented **in an order that I would immediately search by**."

Table 7: Free-form suggestions for question "I do not like the provided function because ...".

I think the provided function can be **improved** by ...	
`scroll`	"It would be nice if the **options displayed in the list respected what you had in the answer line**. For example, I type "update" and all the ones which aren't that disappear." "Have **the top 5 candidates be attached to the scrollable list** function, but frozen as the top 5 results (in the same way you can **freeze rows or columns in excel**)."
`autocomplete`	"Maybe the ability to **hide the top 5 suggestions or hide specific options** would be nice for some users."
`search`	"It would be nice **if the list were a little more readable**, especially when it constantly **changes the boldness/underlines**."
`search-keywords`	"If we could add **more suggest words** to help us find the answer even faster." "the ability to **pull "must include" elements of the utterance into** the initial set of **suggested keywords** would be helpful." "the main thing that would make it work better is just **the model being improved** so predictions are better in general." "**the way matched terms** are both bolded and underlined changes the list of predictions in a way that **sometimes confuses the eyes as you're matching things**."

Table 8: Free-form suggestions for question "I think provided function can be improved by ...".

I would like to have some **other functions** such as ...	
`scroll`	"... **change suggestions** based on the text in the annotation box." "**the displayed list respecting the contents** of the type-box would be nice."
`autocomplete`	"A **searchable dictionary** or **index listing common translation**."
`search`	"An accompanying **lexicon or index of common translations**."
`search-keywords`	"A method of **manually filtering out words** ..."

Table 9: Free-form suggestions for question "I would like to have some other functions such as ...".