Natural Language Decomposition and Interpretation of Complex Utterances

Harsh Jhamtani, **Hao Fang**, **Patrick Xia**, **Eran Levy**,
Jacob Andreas and **Benjamin Van Durme**
Microsoft
{hjhamtani,hafang,patrickxia,erlevy,jaandrea,ben.vandurme}@microsoft.com

Abstract

Designing natural language interfaces has historically required collecting supervised data to translate user requests into carefully designed intent representations. This requires enumerating and labeling a long tail of user requests, which is challenging. At the same time, large language models (LLMs) encode knowledge about goals and plans that can help conversational assistants interpret user requests requiring numerous steps to complete. We introduce an approach to handle complex-intent-bearing utterances from a user via a process of hierarchical natural language decomposition and interpretation. Our approach uses a pre-trained language model to decompose a complex utterance into a sequence of simpler natural language steps and interprets each step using the language-to-program model designed for the interface. To test our approach, we collect and release *DeCU*—a new NL-to-program benchmark to evaluate **De**composition of **C**omplex **U**tterances. Experiments show that the proposed approach enables the interpretation of complex utterances with almost no complex training data, while outperforming standard few-shot prompting approaches.

1 Introduction

Neural sequence models, pre-trained on large datasets of language and code, are extremely effective at parsing natural commands into programs, database queries, and other structured representations of user intent [Chen *et al.*, 2021; Li *et al.*, 2021; Shin *et al.*, 2021; Roy *et al.*, 2022]. However, developing an interface that enables a user to interact with a new API or software system still requires substantial system-specific data collection. Users, meanwhile, may not be aware of the scope of this data collection, and pursue an open-ended set of goals more complicated than those anticipated by system designers.

In this paper, we present DECINT[1], an approach to **dec**ompose complex utterances into a sequence of simpler NL steps, each resembling a simpler elementary utterance that an existing language-to-program **int**erpreter for the NL interface can parse to a sub-program. Consider the utterance *"Exchange the timing of my meetings with Jane and Smith"* (Figure 1). DECINT breaks the utterance down into four NL steps, using a pre-trained LLM and just a few annotated decompositions. The generated NL steps are parsed into programs, relying primarily on a relevant (to the step being parsed) subset of a larger set of existing elementary utterances associated with simpler programs in the target representation. DECINT thus enables an NL interface system to handle user requests representing complex goals (never seen by a semantic parser) by breaking them into a series of NL steps that are interpreted into APIs (never seen by an LLM). Our work is related to recent work which demonstrates that large language models (LLMs) encode knowledge that can be used to interpret complex user goals requiring numerous steps to complete, in setups such as question answering [Wolfson *et al.*, 2020; Khot *et al.*, 2022] and embodied agents [Ahn *et al.*, 2022; Huang *et al.*, 2022]. Compared to such past work, we are concerned with generating programs in a carefully designed intent representation. Starting with labeled elementary utterances, we wish to be able to parse complex utterances that are broader in scope compared to the abundant elementary utterances.

To study utterance decomposition in the NL-to-program space, we collect and release *DeCU*—a new benchmark dataset to evaluate models for **De**composition of **C**omplex **U**tterance. *DeCU* consists of (1) a set of elementary utterances and corresponding programs for managing calendar events and emails and (2) a diverse set of complex user utterances annotated with decompositions into sequences of elementary utterances and their corresponding program fragments. Experiments on *DeCU* show that DECINT outperforms direct few-shot prompting approaches, making it possible to build NL interfaces that accomplish complex goals without large amounts of complex labeled data.

2 Task Overview

We study the problem of parsing a user utterance x into a program y that correctly reflects user intent (Figure 1). We focus on a version of the problem with the following characteristics:

- A domain developer has already collected a dataset of **elementary utterances** annotated with corresponding programs. These utterances represent narrow user goals associated with simple and short programs.

[1]Code and *DeCU* dataset will be available at https://github.com/microsoft/decomposition-of-complex-utterances

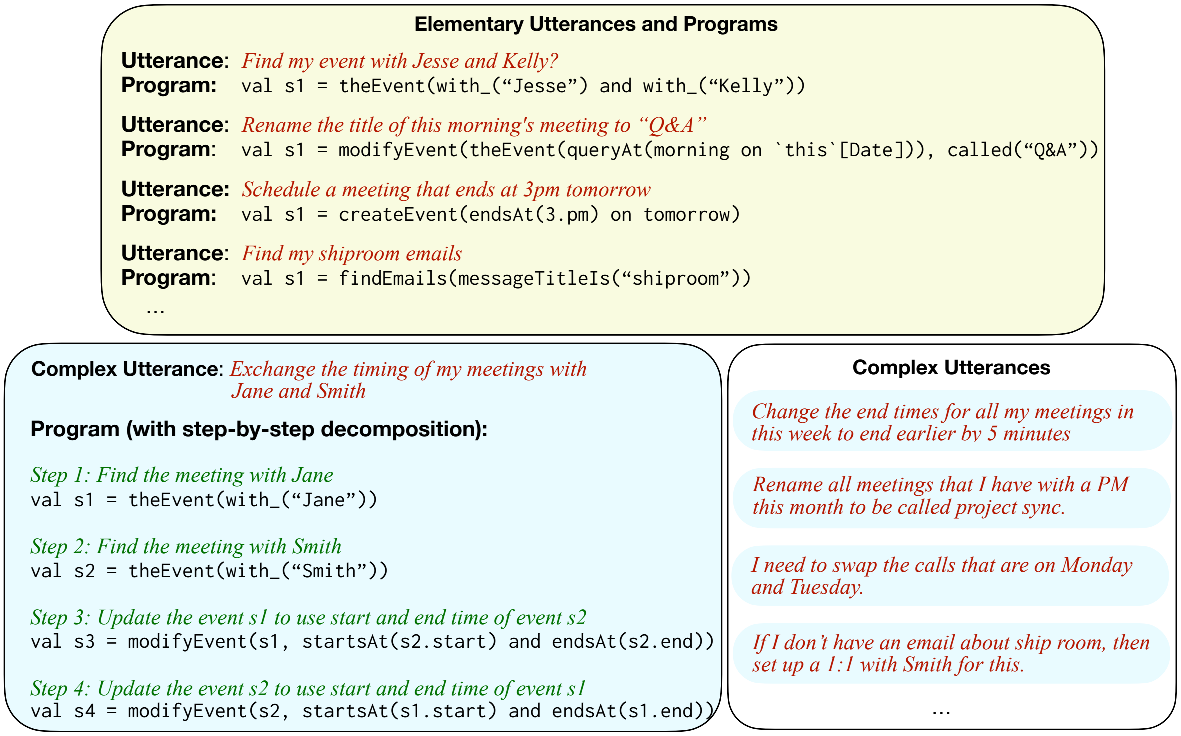

Figure 1: Parsing NL user utterances into programs. We study a scenario in which a large number of elementary utterances have been annotated with programs (top block), and we wish to build a model that can generalize to complex utterances (bottom blocks) requiring more elaborate programs. We introduce a method called DECINT that uses an LLM to decompose a *complex utterance* by predicting *simpler NL steps*, each of which is parsed to a `program` according to the annotated elementary utterances.

- At test time, the system must interpret **complex utterances**. Such utterances require longer programs representing much broader user goals.
- For a small number of complex utterances, we have access to annotations consisting of both natural language decompositions into elementary utterances, and program annotations for elementary utterances.

Annotated complex utterances will in general cover only a small part of the space of possible user requests, and our goal is to build a language-to-program model that can generalize to requests of very different kinds (Figure 1).

3 Data

Many existing relevant decomposition datasets focus on open-ended QA [Wolfson *et al.*, 2020; Khot *et al.*, 2021; Khot *et al.*, 2022; Yang *et al.*, 2018] or robotics domains with a relatively small number of fixed allowed actions [Puig *et al.*, 2018; Shridhar *et al.*, 2020]. By contrast, we are interested in the task of parsing a user utterance to a program that represents the actions to be taken by the interface, grounded on a large number of fixed APIs. Moreover, we want to study how complex user utterances can be supported by the NL interface, without collecting a large amount of additional labeled data, by using decomposition in NL space. To study such multi-step complex intent decomposition, we introduce a new dataset we call *DeCU* (**De**composition of **C**omplex **U**tterances).

The utterances in *DeCU* focus on calendar events and emails. The dataset contains both elementary utterances and complex utterances. Elementary utterances (§3.2) are paired with declarative Scala3 programs based on a domain library (§3.1) that admits a fixed set of APIs and specified types. Complex utterances (§3.3) are annotated with a corresponding sequence of elementary utterances, each paired with a program. Only a few of these complex utterances are included in the training set; they are mainly used to form a test set.

Figure 1 illustrates an example: *"Exchange the timing of my meetings with Jane and Smith"*. How such an utterance should be decomposed is domain-dependent: here, the calendar API does not provide a single endpoint that can swap pairs of meetings; instead, the system must search for the two meetings individually, then update each of their times. Figure 1 shows a possible decomposition into four steps. The first generated NL step, *"Find the meeting with Jane"*, is translated to a program fragment: `val s1 = theEvent(with_("Jane"))`. Individual steps typically represent easier-to-solve inputs for the NL-to-program parser that primarily relies on the annotated elementary utterances.

In addition to domain-specific knowledge of APIs, decomposition of complex utterances often relies on domain-general reasoning and common sense knowledge – for example, to avoid double-counting meetings that match two search results (Figure 2, utterance 1), or to recognize that meetings cannot conflict with themselves (utterance 2).

Utterance 1:	*Change my meetings with Abby and those with Dan this week to start 5 minutes later.*

Decomposition:

```
Step 1: Find events with Abby this week
val s1 = findEvents(with_("Abby") and queryAt(‘this‘[Interval[Date]] and isWeek))
Step 2: Find events with Dan and without Abby this week
val s2 = findEvents(with_("Dan") and not(with_("Abby")) and queryAt(‘this‘[Interval[Date]] and
    isWeek))
Step 3: Set all meetings from the list of events s1 to start 5 minutes later
val s3 = s1.map((x: Event) => modifyEvent(x, startsAt(x.start.local.time + 5.minutes)))
Step 4: Set all meetings from the list of events s2 to start 5 minutes later
val s4 = s2.map((x: Event) => modifyEvent(x, startsAt(x.start.local.time + 5.minutes)))
```

Utterance 2:	*Decline any meeting invitations that are scheduled during my weekly team meeting.*

Decomposition:

```
Step 1: Find the event called "team meeting" that recurs weekly.
val s1 = theEvent(called("team meeting") and recurringWeekly)
Step 2: Find all events.
val s2 = findEvents0
Step 3: Filter events from list s2 to only include ones that intersect with event s1 that are
    not s1.
val s3 = s2.filter((x: Event) => x.interval.intersects(s1.interval) && x.id != s1.id)
Step 4: Decline events in the list s3.
val s4 = s3.map((x: Event) => respond(x, ResponseStatusType.declined))
```

Figure 2: Examples of complex utterances in *DeCU*. Each utterance is accompanied by decompositions consisting of a sequence of NL steps and associated program fragments, annotated by domain experts.

3.1 Domain Library

The domain library defines the set of types and functions available for program annotations. Types model objects such as `Person` and `Event`, whereas functions represent actions that can be taken by the agent, including high-level APIs (*e.g.,* `createEvent`, `findEmails`), low-level operations (*e.g.,* `min`, `+`), predicate constructors (*e.g.,* `called`, `startsAt`), etc. The domain library for *DeCU* is packaged as standard Scala source code, consisting of 33 types and over 200 functions.[2]

3.2 Elementary Utterances

DeCU contains 841 elementary utterances paired with programs. A few examples are shown in the top box in Figure 1. These utterances are *elementary* in that they represent narrow user goals such as creating or deleting a single meeting, which can typically be achieved using a single API. As such, they have relatively short programs, generally less than 5 tokens.[3] Examples are written and reviewed by domain experts who are familiar with the domain library (on account of their experience from working with a deployed system leveraging such a library) and annotation guidelines.

3.3 Complex Utterances

To study how *complex* utterances can be supported by an NL interface, we collect a diverse set of more involved user requests, and annotate these with decompositions into elementary steps, along with programs for each step. As the name suggests, compared to elementary utterances, these utterances represent more complex and broader user goals, with the corresponding programs typically being much longer (an average of 14.5 tokens per program). To collect complex utterances, we employ a mix of manual authoring and automated utterance generation. Manual authoring is performed by domain experts with a focus on diversity and goals that require the composition of multiple calls to the domain APIs. For automated collection techniques, we generate utterances using GPT-3 [Brown *et al.*, 2020] prompted with a few random examples of manually-authored utterances. About 60% of all the collected utterances were generated automatically. Appendix A[4] provides more details on utterance collection. Examples are shown in Figure 1.

Decomposition Annotations: Six annotators familiar with the domain (annotators had past experience working with the domain library) decompose complex utterances into elementary ones. When results from earlier steps must be reused, these NL decompositions may include explicit reference to earlier step outputs (Figure 2). More information about annotator instruction is provided in Appendix A. Each annotation was additionally reviewed by two additional domain experts, separate from the set of 6 annotators.

Data Statistics: We collected a total of 210 unique complex utterances. The dataset is a mix of 126 utterances paired with annotated programs and 84 that are unannotated. As discussed later, in addition to reference-based metrics, we also provide various reference-less metrics that do not require annotations. While it is a relatively small count, note that most of the data (200 out of 210) is used to construct an evaluation set, as we are interested in learning to generalize from very small numbers of training examples. Additionally, we would like to note that our dataset is of similar scale as some other recent datasets: SayCan [Ahn *et al.*, 2022] was evaluated only on 101 examples, and each of the Big-Bench [Suzgun *et al.*, 2022] hard task used less than 250 examples for evaluation. Annotated complex utterances in our full dataset exhibit a

[2] Some built-in types (*e.g.,* `String`, `Boolean`), functions (*e.g.,* `map`), and control flow statements (*e.g.,* `if`) are not explicitly defined and counted. Appendix B provides more details.

[3] To compute this statistic, programs are split into tokens based on heuristics, treating API names, argument names, and values as individual tokens.

[4] Appendix available at https://github.com/microsoft/decomposition-of-complex-utterances

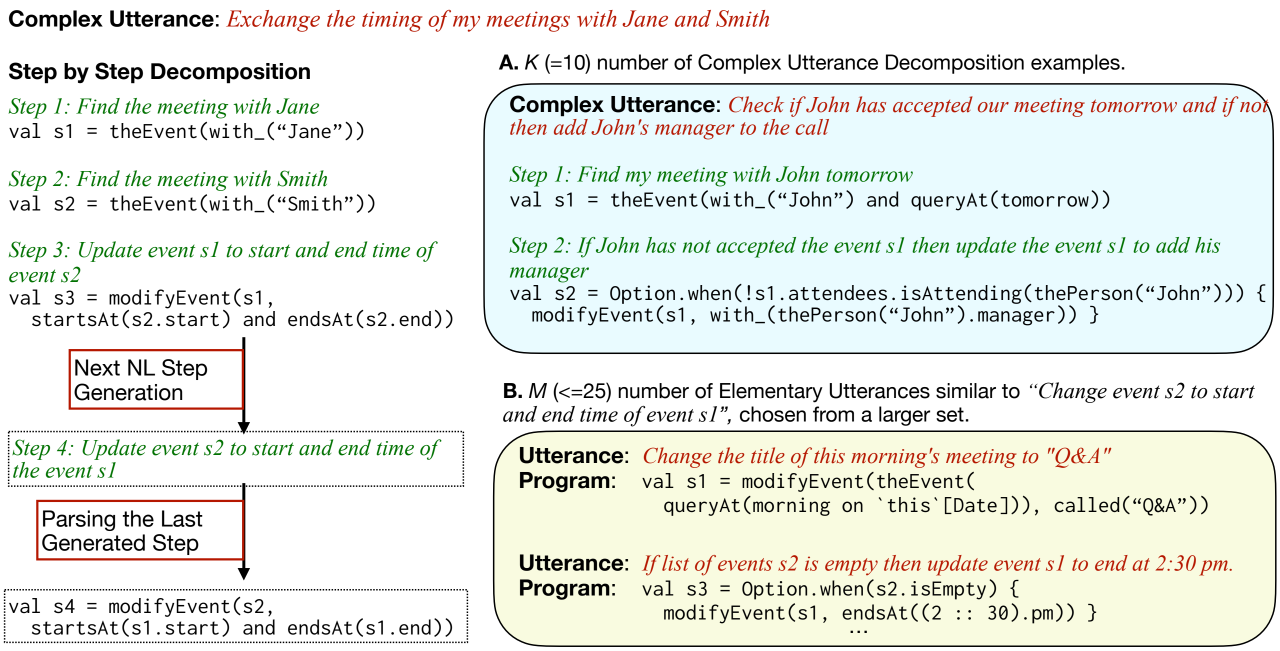

Figure 3: DECINT maps complex utterances into elementary steps, each of which is parsed in sequence to arrive at a final program. NL decomposition and program generation steps are interleaved. While parsing a step, up to M similar examples of elementary utterances are retrieved.

diverse range of properties (an utterance can have multiple): 55% use a map operation (for-loop), 36% contain actions based on a condition, 31% use a filter operation, 24% query about calendar/email, 37% contain a create meeting action, 9% contain a delete meeting action, and 31% contain a modify meeting action. The average number of decomposition steps in our data is 3, with a maximum of 7 steps. The average number of tokens in each program is 14.5[5], while the average number of tokens in the program fragment corresponding to a single step is 4.8. For comparison, the average number of tokens in the programs for elementary utterances is 4.5.

4 Approach

The DECINT approach, illustrated in Figure 3, maps a complex utterance x to a sequence of interpretable lower-level NL steps $(z_1, z_2, ..)$ that resemble elementary utterances. Each step or low-level utterance z_j is parsed into a program fragment y_j. In particular, DECINT maps from commands to programs according to the following iterative generative process:

1. **Natural Language Decomposition**:
   $z_j \sim p(\cdot \mid x, z_{<j}, y_{<j})$.
2. **Program Generation**:
   $y_j \sim p(\cdot \mid x, z_{\leq j}, y_{<j})$.

NL Decomposition (§4.1) and program generation (§4.2) steps are interleaved, with later portions of the language decomposition conditioned on earlier program fragments. In principle, one could also condition on the return values of the earlier program fragments (see Limitations section). We do not do so in this paper, as running the programs would require API implementations and input data.

4.1 Natural Language Decomposition

The NL decomposition stage generates the next NL step z_j conditioned on the user utterance x and any previously generated steps and program fragments. We obtain z_j by greedy decoding from a pre-trained LLM in a few-shot in-context learning setup [Brown *et al.*, 2020]. The model is prompted with $K = 10$ example decompositions, each of which consists of an utterance x followed by any previous steps and their program fragments, all concatenated together $(x, z_1, y_1, z_2, , ..., z_N, y_N)$. We additionally found it useful to include a list of up to M elementary NL utterances at the start of the prompt (before the K decomposition examples), selecting the ones with highest BM25 similarity to the input utterance. This is intended to inform the model about the kind of elementary steps the NL-to-program parser can handle. (An example constructed prompt is shown in Appendix C.) Example decompositions are taken from the set of 10 complex utterances in the training split of *DeCU*.

DECINT's ability to perform NL decomposition thus results from a combination of parametric knowledge about the structure of programs in general (the result of pretraining) and non-parametric knowledge about the domain of interest (obtained via in-context learning). Together, these enable generalization to structurally novel user requests. For example, there are no training examples that involve exchanging the timing of two meetings (the test example in Figure 3), but DECINT nonetheless synthesizes a correct program.

4.2 Program Generation

The program generation step synthesizes a program fragment y_j for a given NL step z_j, conditioned on any preceding steps and incomplete program. This is a well-studied semantic parsing problem, and we design the NL-to-program parser largely following past work that applies pre-trained LLMs. We use in-context learning with dynamically selected prompt examples from the set of elementary examples data [Brown *et al.*, 2020]. As before, we use greedy decoding. Generated program fragments may refer to previously generated fragments using named step variables. For a given NL utterance or step, we identify up to M examples from the set of elementary utterances, where each example is an (utterance, program) pair (as shown in box B in Figure 3). The selection of the examples is based on the similarity of the utterance to the intermediate NL step being parsed. To compute similarity, we again use BM25, as in past work [Rubin *et al.*, 2022; Roy *et al.*, 2022]. In pilot experiments on training data, we discovered it was useful to also include the K decomposition examples at the bottom of the prompt (detailed prompt example shown in Appendix C). This may be because the decomposition examples provide a demonstration of how to generate program fragments for a step conditioned on previous steps and help bridge any possible domain shift from elementary to complex utterances.

4.3 Baselines

The DECINT method decomposes a complex utterance into NL steps, separately parsing each step, and using internal variable references to assemble a larger program. The standard few-shot prompting approach for tasks like this one instead directly predicts the parse without generating the intermediate NL steps [Roy *et al.*, 2022]. We compare to this approach, which we denote **DIRECT-PRED**, in our experiments. There are a few key differences compared to the DECINT method. Complex utterance examples are presented without the intermediate NL steps (*i.e.,* each utterance is paired with a multi-line program). The output generation is a single-step process since there are no intermediate NL steps that need to be generated. As with DECINT, examples of elementary utterances are also included in the prompt. We also consider a **COT** [Wei *et al.*, 2022] baseline, wherein the model first predicts all intermediate NL steps and then predicts the program. Accordingly, the complex utterance examples in the prompt are annotated with intermediate steps. This baseline resembles the method proposed in Jiang et al [2023]. Note that compared to COT, DECINT interleaves step generation and parsing, and dynamically updates the subset of exemplars from elementary utterances to be relevant to the step being parsed.

We also report results using a variant of DECINT that relies only on K decomposition exemplars but without access to elementary utterances (M=0 instead of 25). We refer to such a baseline as **FEW-SHOT**. We also consider a variant of DECINT that uses only a single decomposition exemplar (K=1 instead of 10), and thus relies almost entirely on the elementary utterances from the underlying domain. We refer to the variant as **ELEMENTARY-ONLY**. Finally, we also report results on a variant of DECINT that employs a fixed set (randomly selected) of elementary utterance exemplars. We refer to this variant as **REACT**, as similar to ReAct [Yao *et al.*, 2023], it does not employ any dynamic exemplar selection. This is in contrast to DECINT that dynamically identifies elementary utterances most similar to the generated step being parsed.

5 Experiments

5.1 Evaluation

Overlap with Reference Programs: We report Exact Match (**EM**) and character-based edit distance (**CER**) metrics[6] against the gold program. Before computing these metrics, we normalize the programs by lowercasing the entire program and removing extra spaces. Since there can be multiple possible ways to express the target multi-line program, Exact Match can only be viewed as a lower-bound metric for this task. These metrics are reported only for the subset of the data that consists of annotated reference programs.

Well-formed Evaluation: Additionally, we report the fraction of predictions that are valid (**WellForm**) under the domain library, *i.e.,* the full program follows correct syntax and only uses functions available in the library. Note that WellForm does not necessarily represent correctness with respect to the user goal. We report the metric for the entire test set.

Program Correctness: Finally, we report the overall correctness of the generated programs. We define a program to be correct overall if: it is well-formed, and correctly represents the user request. We use GPT-4 (gpt-4-32k) [OpenAI, 2023] to rate the correctness of the generated programs (**Correct**). The prompt consists of an instruction and four manually labeled exemplars (two "correct" and two "incorrect") followed by the test example. Each example is a user utterance followed by the associated program. The label is a natural language caption/explanation of the generated program, followed by a final verdict on whether the generated program is "correct" or "incorrect" for the given user utterance – following a chain-of-thought style prediction[7]. Since we have an automatic static analysis to infer exactly which programs are well-formed (WellForm), outputs that are not well-formed are automatically considered to be incorrect as per the definition above (but are included in the denominator for all evaluations). Note that the Correct metric is reference-less, is easier to scale than human evaluations, and correlates well with human ratings (Section 5.3).

5.2 Setup

We consider the task of parsing complex utterances in *DeCU* given only ten complex utterances (annotated with decompositions) to be used as training data (exemplars for in-context learning). We report results on the test set consisting of the remaining 200 complex utterances. We use a maximum of $M \leq 25$ additional elementary utterances (as many as permitted by the LM's context window) selected according to BM25 similarity with the step being parsed. We use OpenAI's *text-davinci-003* model as the LLM for generating each NL step as well as for parsing it into a program.

[6] https://huggingface.co/spaces/evaluate-metric/cer

[7] The exact prompts used in Correct are presented in Appendix D

Utterance 1: *Make me a meeting at 3 PM mountain time and refuse any meetings that happen at the same time.*

DECINT

```
Step 1: Find events at 3 pm mountain time
val s1 = findEvents(queryAt(3.pm inZone TimeZone("MST")))
Step 2: Refuse any events in the list s1
val s2 = s1.map(x => respond(x, withResponse(ResponseStatusType.declined)))
Step 3: Create a meeting at 3 pm mountain time
val s3 = createEvent(createAt(3.pm inZone TimeZone("MST")))
```

DIRECT-PRED

```
val s1 = createEvent(createAt(3.pm inZone TimeZone("MST")))
val s2 = findEvents(queryAt(3.pm inZone TimeZone("MST")))
val s3 = s2.map(x => refuseEvent(x))
```

COT

```
Step 1: Create a meeting at 3 pm mountain time
Step 2: Refuse any meetings that happen at the same time"}
val s1 = createEvent(createAt(3.pm inZone TimeZone(\"MST\")))
val s2 = findEvents(queryAt(3.pm inZone TimeZone(\"MST\"))).map(x => refuseEvent(x))
```

Comment We observe that hallucination of functions is much more prominent in the baselines DIRECT-PRED and COT, as can be seen in the use of the non-existing function `refuseEvent`

Utterance 2: *I'd like to divide the meeting at noon in two, so split off the last fifteen minutes as a separate event.*

DECINT

```
Step 1: Find the event at noon
val s1 = theEvent(createAt(noon))
Step 2: Subtract 15 minutes from the end time of the event s1
val s2 = s1.end - 15.minutes
Step 3: Update the event s1 to end at the time instant s2
val s3 = modifyEvent(s1, endsAt(s2))
Step 4: Create a new event starting at the time instant s2
val s4 = createEvent(startsAt(s2))
```

Comment Output program from DECINT was judged as incorrect since it doesn't specify the duration of the second event.

Figure 4: Outputs from DECINT and baselines on sample complex utterances in *DeCU*.

System	Correct↑	WellForm↑	EM↑	CER↓
DIRECT-PRED	0.34	0.36	0.04	0.44
COT	0.25	0.29	0.05	0.46
FEW-SHOT	0.13	0.19	0.00	0.50
REACT	0.21	0.23	0.02	0.49
ELEMENTARY-ONLY	0.23	0.31	0.04	0.54
DECINT	**0.41**	**0.46**	0.05	**0.40**

Table 1: Quality of the generated program for complex utterances under various automated metrics.

5.3 Evaluation of Generated Program

Table 1 reports various automated metrics. DECINT outperforms all the baselines, sometimes by a wide margin. As can be seen in the table, DECINT outputs receive an overall correctness score (**Correct**) of 41% for complex utterances compared to 34% and 25% for the baselines DIRECT-PRED and COT respectively.[8] We posit that DECINT is able to make more effective use of pretraining by breaking down a complex command into NL steps and retrieving relevant exemplars for each step. Further, FEW-SHOT, that is equivalent to DECINT with M=0, fares badly, suggesting that DECINT relies on information from elementary utterances in addition to supervised decompositions. Finally, ELEMENTARY-ONLY, which is equivalent to DECINT with K=1, also does worse than DECINT, suggesting the usefulness of a handful of supervised decompositions. Note, however, that a 54% of the predictions from DECINT are not well-formed, indicating that even structural generalization in *DeCU* remains a major challenge. Nonetheless, DECINT fares better compared to other methods on WellForm metric.

Human Evaluation for Program Correctness: We also obtained the overall program correctness rating ("correct" vs "incorrect" for a user utterance) from human evaluators familiar with the domain library. Just as was the case with **Correct** metric, outputs that are not well-formed are automatically considered incorrect. The aggregate scores for DECINT, DIRECT-PRED and COT (our method and the two top performing baselines as per automated **Correct** metric) under human evaluation are 41%, 33% and 26% respectively, which are very close to the scores for these methods under the automated **Correct** metric. Additionally, we observe a high correlation between human annotator-provided judgment and **Correct** judgments (a more detailed correlation analysis is provided in the Appendix D).

Results with other LLMs: We also report results using GPT-4 (*gpt-4-32k*) and LLAMA-2-70B [Touvron *et al.*, 2023] as the underlying LLM. Due to cost considerations, we report results only for the top three methods from Table 1. We observe that DECINT outperforms the baselines, demonstrating our approach is effective across underlying LLMs (Table 2).

5.4 Evaluation of NL Decomposition

We measure whether the NL decomposition steps altogether are *sufficient and correct* to complete the user request.[9] For

[8] Differences are significant ($p < 5\%$) using bootstrap resampling.

[9] Unless stated otherwise, all analysis uses outputs with *text-davinci-003* as the underlying LLM

System	Correct↑/ WellForm↑ GPT-4	LLAMA2-70B
DIRECT-PRED	0.35 / 0.39	0.25 / 0.41
COT	0.37 / 0.40	0.23 / 0.32
DECINT	**0.49 / 0.56**	**0.35 / 0.50**

Table 2: Results using GPT-4/LLAMA-2-70B as the LLM.

example, the output from DECINT for the second utterance in Figure 4 is *not* sufficient and correct because the fourth step fails to specify the duration of the meeting, which is supposed to be 15 minutes as per user request. A random subset of 40 of DECINT NL predictions and corresponding expert annotations were manually labeled by one of the authors as correct or incorrect. The expert annotations and DECINT predictions were rated as 98% and 85% correct, respectively. Future work can explore ways to further improve the accuracy of the predicted NL steps. We also conducted a step-level evaluation, which we discuss in Appendix D.

5.5 Qualitative Analysis

We provide example predictions in Figure 4, with additional examples provided in the Appendix. Additionally, we perform an error analysis of the NL-to-program step of DECINT. We restrict the study to the predictions that were labeled as incorrect in Table 1. The most common issues are those that make the program not well-formed, as summarized in Table 1. Many errors are due to *nonexistent APIs / API arguments* (21% of the incorrect programs have at least this problem) and *nonexistent type attribute* (43%). A smaller number result from even more basic *syntax errors* and *type mismatches* (17%). Future work could constrain the outputs of the parser [Shin *et al.*, 2021] to only use allowed functions and follow correct syntax, though such approaches can substantially increase the cost of decoding.

A few errors result from predictions that capture only *partial user intent* (6%). For example, for utterance 2 in Figure 4, the prediction does not capture the user intent of creating the second event for 15 minutes. Many of the remaining errors involve more fundamental semantic mismatches between user intents and model outputs. For example, for *"Loop around all my 1/1 meetings this week so that they also happen next week"*, the prediction updates the meetings this week instead of creating another set of meetings next week.

6 Related Work

Past work has explored using command decomposition to break down complex tasks or requests into smaller subtasks that are easier to manage. The LaMDA model [Thoppilan *et al.*, 2022], for example, is capable of breaking down "How to" type queries into steps. However, generated steps are not tied to any actions or APIs, and are more in the form of a narrative rather than executable steps.

Khot et al. [2021] decompose a question into sub-questions that can be answered by a neural factoid single-span QA model and a symbolic calculator. Drozdov et al. [2022] decompose an utterance using a syntactic parse. However, not all utterances in our dataset would lend to such a style of decomposition, since all required actions might not align to a part of the parse. Recent work [Jiang *et al.*, 2023] has also explored first generating an entire plan in NL and then generating a program. Paranjape et al. [2023] focus on using tools and python scripts to complete a given task such 'Translate into Pig Latin'. Compared to such past work, the complex utterances in our case are decomposed into intermediate steps that are parsed into a sub-program in the target representation as opposed to generating Python programs. Additionally, these sub-programs are a part of the final program output and thus we care about the accuracy of intermediate steps as well.

A related area of research involves grounding high-level tasks, expressed in natural language, to a chosen set of actionable steps that a robot could take [Sharma *et al.*, 2022; Singh *et al.*, 2022; Ahn *et al.*, 2022; Huang *et al.*, 2022]. Huang et al [2022] propose a method to ground high-level tasks such as 'make breakfast' to a set of actionable steps such as 'open fridge'. Such work typically assumes a fixed inventory of low-level actions. For example, 'Semantic Translation' discussed in Huang et al [2022] translates the predicted step into an admissible action by calculating the semantic distance of the predicted action phrase against *all* possible actions. The APIs in our case can be composed and chained together, and have optional arguments. So identifying an exhaustive set of allowed actions under the DSL (domain-specific-language) in question is intractable.

7 Conclusion

We have presented DECINT, an approach for interpreting complex user utterances by decomposing them into elementary natural language steps. To evaluate methods for generating programs from user requests, we have introduced the *DeCU* dataset, featuring a diverse set of utterances requiring substantial generalization from a small training set. Experiments on *DeCU* show that DECINT outperforms a standard few-shot prompting approach to program generation, with additional analysis revealing opportunities for improvement in both natural language decomposition and program generation phases.

Ethical Statement

We leverage pre-trained neural language models such as GPT-3, and systems built using our approach might inherit some biases present in these pre-trained models. We build a system for NL-to-program, that users can leverage to command various NL interfaces. Such systems are not perfectly accurate and should be carefully deployed since they may lead to unintended side effects.

Acknowledgements

We would like to thank Jason Eisner, Subhro Roy, Jason Wolfe, Matt Gardner, and Val Ramirez for valuable discussions. We thank anonymous reviewers for feedback on the paper.

References

[Ahn *et al.*, 2022] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J. Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, and Mengyuan Yan. Do as I can, not as I say: Grounding language in robotic affordances. *arXiv: 2204.01691*, 2022.

[Brown *et al.*, 2020] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[Chen *et al.*, 2021] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv: 2107.03374*, 2021.

[Drozdov *et al.*, 2022] Andrew Drozdov, Nathanael Schärli, Ekin Akyürek, Nathan Scales, Xinying Song, Xinyun Chen, Olivier Bousquet, and Denny Zhou. Compositional semantic parsing with large language models. *arXiv: 2209.15003*, 2022.

[Huang *et al.*, 2022] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 9118–9147. PMLR, 17–23 Jul 2022.

[Jiang *et al.*, 2023] Xue Jiang, Yihong Dong, Lecheng Wang, Qiwei Shang, and Ge Li. Self-planning code generation with large language model. *arXiv: 2303.06689*, 2023.

[Khot *et al.*, 2021] Tushar Khot, Daniel Khashabi, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Text modular networks: Learning to decompose tasks in the language of existing models. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 1264–1279. Association for Computational Linguistics, 2021.

[Khot *et al.*, 2022] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. *arXiv: 2210.02406*, 2022.

[Li *et al.*, 2021] Haoran Li, Abhinav Arora, Shuohui Chen, Anchit Gupta, Sonal Gupta, and Yashar Mehdad. MTOP: A comprehensive multilingual task-oriented semantic parsing benchmark. In Paola Merlo, Jörg Tiedemann, and Reut Tsarfaty, editors, *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*, pages 2950–2962. Association for Computational Linguistics, 2021.

[OpenAI, 2023] OpenAI. Gpt-4 technical report, 2023.

[Paranjape *et al.*, 2023] Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*, 2023.

[Puig *et al.*, 2018] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8494–8502, 2018.

[Roy *et al.*, 2022] Subhro Roy, Sam Thomson, Tongfei Chen, Richard Shin, Adam Pauls, Jason Eisner, and Benjamin Van Durme. BenchCLAMP: A benchmark for evaluating language models on semantic parsing. *arXiv: 2206.10668*, 2022.

[Rubin *et al.*, 2022] Ohad Rubin, Jonathan Herzig, and Jonathan Berant. Learning to retrieve prompts for in-context learning. In Marine Carpuat, Marie-Catherine de Marneffe, and Iván Vladimir Meza Ruíz, editors, *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pages 2655–2671. Association for Computational Linguistics, 2022.

[Sharma *et al.*, 2022] Pratyusha Sharma, Antonio Torralba, and Jacob Andreas. Skill induction and planning with latent language. In *Proceedings of the Annual Association for Computational Linguistics*, 2022.

[Shin *et al.*, 2021] Richard Shin, Christopher H. Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. Constrained language models yield few-shot semantic parsers. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 7699–7715. Association for Computational Linguistics, 2021.

[Shridhar *et al.*, 2020] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020.

[Singh *et al.*, 2022] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv: 2209.11302*, 2022.

[Suzgun *et al.*, 2022] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

[Thoppilan *et al.*, 2022] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, YaGuang Li, Hongrae Lee, Huaixiu Steven Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Yanqi Zhou, Chung-Ching Chang, Igor Krivokon, Will Rusch, Marc Pickett, Kathleen S. Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Soraker, Ben Zevenbergen, Vinodkumar Prabhakaran, Mark Diaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravi Rajakumar, Alena Butryna, Matthew Lamm, Viktoriya Kuzmina, Joe Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Agüera y Arcas, Claire Cui, Marian Croak, Ed H. Chi, and Quoc Le. Lamda: Language models for dialog applications. *CoRR*, abs/2201.08239, 2022.

[Touvron *et al.*, 2023] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[Wei *et al.*, 2022] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[Wolfson *et al.*, 2020] Tomer Wolfson, Mor Geva, Ankit Gupta, Yoav Goldberg, Matt Gardner, Daniel Deutch, and Jonathan Berant. Break it down: A question understanding benchmark. *Trans. Assoc. Comput. Linguistics*, 8:183–198, 2020.

[Yang *et al.*, 2018] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

[Yao *et al.*, 2023] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.