Diverse In-Context Example Selection After Decomposing Programs and Aligned Utterances Improves Semantic Parsing

Mayank Kothyari[1*], Sunita Sarawagi[1*], Soumen Chakrabarti[1]
Gaurav Arora[2], Srujana Merugu[2]
[1]Indian Institute of Technology Bombay [2]Amazon

Abstract

LLMs are increasingly used as seq2seq translators from natural language utterances to structured programs, a process called semantic interpretation. Unlike atomic labels or token sequences, programs are naturally represented as abstract syntax trees (ASTs). Such structured representation raises novel issues related to the design and selection of in-context examples (ICEs) presented to the LLM. We focus on *decomposing* the pool of available ICE trees into fragments to minimize interference from irrelevant content and improve generalization on test instances. Next, we propose how to use (additional invocations of) an LLM with prompted syntax constraints to automatically map the fragments to corresponding utterances. Finally, we adapt and extend a recent method for diverse ICE selection to work with whole and fragmented ICE instances. We evaluate our system, SCUD4ICL[1], on popular diverse semantic parsing benchmarks, showing visible accuracy gains from our proposed decomposed diverse demonstration method. Benefits are particularly notable for smaller LLMs, ICE pools having larger labeled trees, and programs in lower resource languages.

1 Introduction

Large language models (LLMs), being proficient program generators (Yan et al., 2023), are well-suited to solving semantic interpretation tasks: translating natural language *utterances* $\mathbf{x}_0$ (which could be questions or instructions) into executable *code* $\mathbf{y}_0$ (interchangeably called program, or query) in a structured language, such as Python, SQL or SPARQL, possibly even invoking libraries (Hsieh et al., 2023) such as PyTorch or Pandas (Ye et al., 2024). Here we characterize code using their abstract syntax trees (ASTs).

Pretraining corpora of LLMs include diverse public schema, structured data and utterances, e.g., in the field of text2sql or knowledge graph question answering (KGQA). When applied to less popular domains like calendar management (Andreas et al., 2020, SMCalFlow) or inspecting geographical databases (Zelle and Mooney, 1996, GeoQuery), pretrained LLMs perform less impressively. This can be a serious impediment to exploiting LLMs for structured interpretation in settings where the schema and data are private, e.g., in case of enterprise data that were not part of the pretraining corpus. Given the enormous size of the best LLMs, continued training or fine-tuning may be impractical for most users.

LLMs are also known to be effective in-context learners (ICL) (Brown et al., 2020; Lu et al., 2023). This capability may be particularly beneficial (Levy et al., 2023) when in-context examples (ICEs) involve the same (possibly private) schema and task. Formally, given test utterance $\mathbf{x}_0$, the input to the LLM includes M in-context examples represented in the form $\boldsymbol{p}; \mathbf{x}_1, \mathbf{y}_1; \ldots; \mathbf{x}_M, \mathbf{y}_M; \mathbf{x}_0$ with suitable delimiters. Here, $\boldsymbol{p}$ is an optional instruction prefix, and the LLM has to decode $\hat{\mathbf{y}}_0$, the translation of $\mathbf{x}_0$. A common challenge for ICL is to select $S = \{(\mathbf{x}_m, \mathbf{y}_m) : m \in [M]\}$[2] from a larger corpus of instances to maximally assist the LLM, the guiding principles being: (1) M should be small to reduce forward inference cost. (2) $\mathbf{x}_m$ should be strongly related to $\mathbf{x}_0$. (3) S should be suitably diverse, in an attempt to provide adequate coverage of possible target ASTs.

Our point of departure is to bring a new desirable criterion into the above picture of ICL. ASTs $\{\mathbf{y}_m : m \in [M]\}$, as well as the target $\mathbf{y}_0$ are complex structured outputs. The ideal $\mathbf{y}_0$ may have only *partial overlap* with ASTs in the ICEs $\{\mathbf{y}_m : m \in [M]\}$, but the non-overlapping parts

*Corresponding authors: maykat2017@gmail.com, sunita@iitb.ac.in

[1]The code and dataset for the paper are available at https://github.com/iMayK/SCUD4ICL

[2]$[M]$ refers to the set $\{1, 2, \ldots M\}$

Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 5611–5629

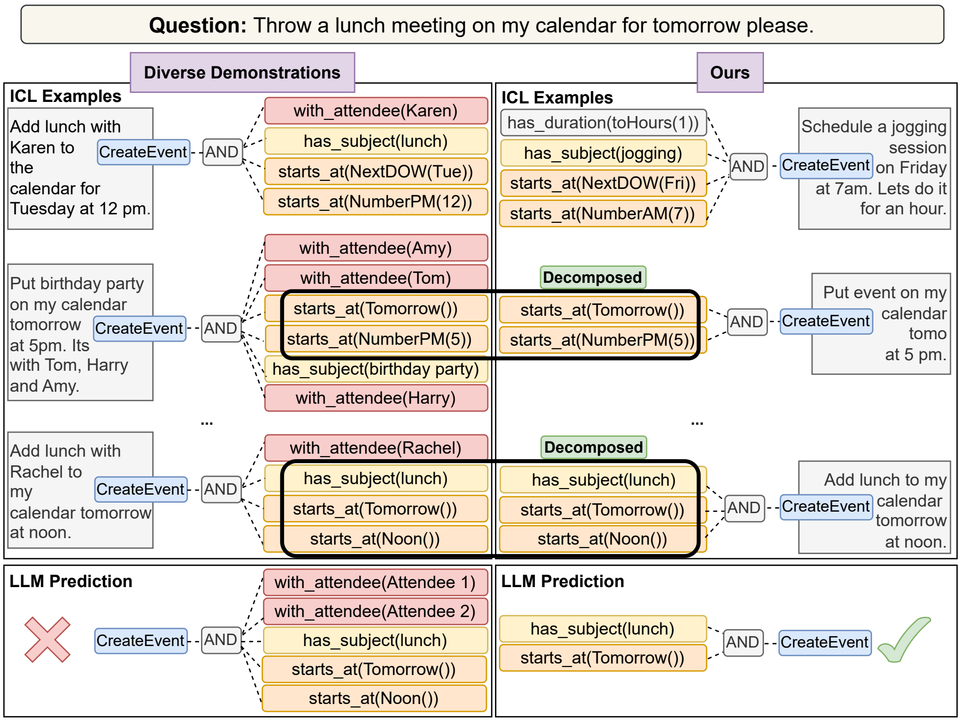

Figure 1: An example of how decomposed queries help avoid interference. On the left are three whole ICEs selected by an existing method. On the right are SCUD4ICL's ICEs. Note that two of these are decompositions of training examples, after removing irrelevant clauses. Removing the irrelevant clauses reduces interference during ICL leading to a correct prediction from the LLM.

may distract in-context learning. This motivates our first contribution: select ICEs from not only 'whole' ASTs, but also *decompose* them into meaningful fragments (typically, subtrees) and make them available for the ICE subset selector. This is in sharp contrast to all existing methods that choose whole examples for ICL demonstration.

The enhancement proposed above raises the issue that even if $(\mathbf{x}_m, \mathbf{y}_m)$ pairs are provided for complete utterances $\mathbf{x}_m$ and corresponding complete ASTs $\mathbf{y}_m$, and if the decomposition of $\mathbf{y}_m$ into useful fragments $\mathbf{y}_{m,k}$ were possible to automate, these AST fragments do not come with corresponding sub-utterances $\mathbf{x}_{m,k}$. Our second contribution is to employ an LLM for this translation task, but with a twist that turns out to be critical: we instruct the LLM to regard the utterances $\mathbf{x}_{m,k}$ to be generated as *sub-utterances* of $\mathbf{x}_m$.

Our third contribution is, given $\mathbf{x}_0$, to select, from the available pool of whole- and fragmented ICEs, a suitable subset to include in the LLM input. This is also a delicate step, because inclusion of ICEs irrelevant to $\mathbf{x}_0$ (including cases with complete ASTs) have the potential (Liu et al., 2023a; Chen et al., 2023) to interfere with the generation of the correct parse $\mathbf{y}_0$. We show examples of such in-context interference for a semantic parsing task in Figure 1. We extend a recent diverse demonstration method (Levy et al., 2023) to handle complete and decomposed ICEs seamlessly.

Our system, SCUD4ICL (sub code+utterance decomposition for in-context learning), incorporates all the three enhancements above, and will be released publicly on acceptance.

We present empirical evaluations with three popular diverse semantic parsing benchmarks: SMCalFlow, GeoQuery, and MTOP. We explore the effects of diverse training sizes, train-test discrepancy, models, decomposition depths, and code languages. Apart from consistent wins at semantic interpretation, we establish that fragmented ICE availability, coupled with our selection criterion, reduces harmful interference. Benefits are particularly notable for smaller LLMs, larger labeled trees, and lower resource languages.

2 Related Work

Our current work on semantic parsing using LLMs with ICL leverages prior work on selection of ICEs and query decomposition techniques.

In-context example selection It is crucial to select ICEs that are highly informative with respect to the test utterance. An intuitive approach is to select

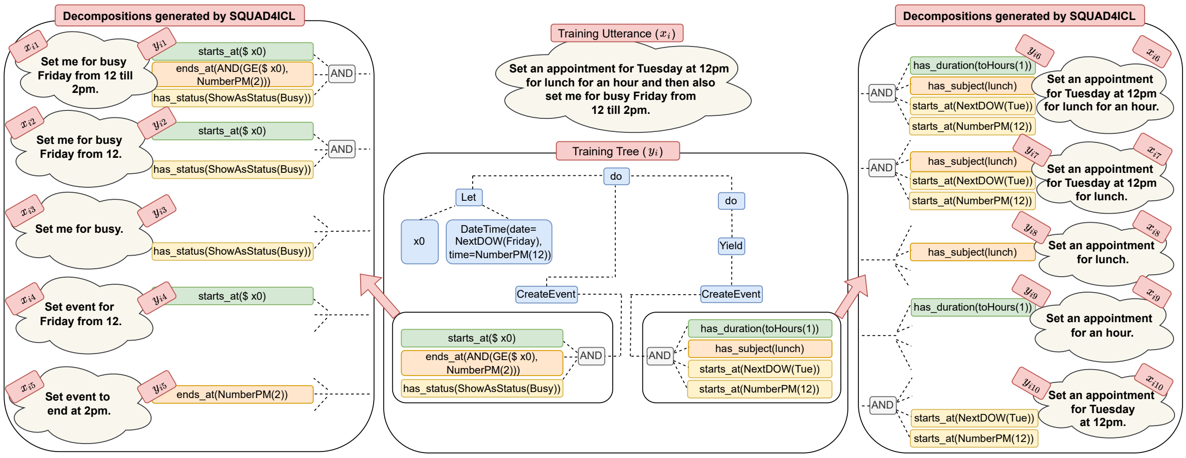

Figure 2: An example showing decomposition of a training instance by SCUD4ICL. A complex training utterance-tree pair ($\mathbf{x}_i$, $\mathbf{y}_i$) comprising of more than ten clauses is decomposed into ten subtrees of varying complexity. The sub-utterances $\mathbf{x}_{i,j}$ attached to each sub-tree $\mathbf{y}_{i,j}$ are subsumed by $\mathbf{x}_i$ while being fluent and relevant to the respective $\mathbf{y}_{i,j}$. The `"Let"` clause, which defines $\mathbf{x}_0$, is repeated in subqueries wherever needed, but we omit repetition in the figure to reduce clutter.

the top-K utterances most similar to the test utterance (Liu et al., 2022; Rubin et al., 2021). However, this method often results in redundancy and limited coverage. Gupta et al. (2023) introduce an unsupervised set-selection approach to mitigate this problem. Similarly, Hongjin et al. (2022) propose an unsupervised, graph-based strategy that combines similarity and diversity. However, their work focuses on reducing annotation of ICL examples, rather than query-specific selection.

More recently, Ye et al. (2023) have formulated ICL example selection as subset selection using Determinantal Point Processes (DPPs) (Kulesza and Taskar, 2011). This approach optimizes example selection through a contrastive learning objective that balances relevance and diversity, using a specially designed kernel. Building on this, Fu et al. (2024) propose TISE, a tripartite selection method that incorporates contextual relevance, event correlation, and example diversity. For semantic parsing applications, where structure encodes vital information, Bogin et al. (2024); Levy et al. (2023) propose generating diverse demonstrations by collectively covering the maximal number of local structures in the test utterance.

All the above methods, however, treat both $\mathbf{x}_m$ and $\mathbf{y}_m$ as monolithic. When the selected ICEs contain irrelevant sub-parts, there is a high risk of the LLM being misled (as we shall establish). While our work builds on the diverse selection method of Levy et al. (2023), we first generate an enhanced pool of ICEs obtained by decomposition. This enables the selection step to protect the LLM from interference from irrelevant fragments.

Example Decomposition In complex question answering (QA) and semantic parsing, the utterance is often decomposed into simpler sub-utterances that are mapped to sub-queries, whose responses are later assembled into the final answer. Liu et al. (2023b) propose a hybrid complex QA system performs top-down parsing of questions into tree-structured representations (referred to as H-expressions). Huang et al. (2023) present a neural model that hierarchically decomposes complex questions into trees. Shi et al. (2023b) propose a novel execution decomposition (ExeDec) strategy for tackling the broader problem of program synthesis by addressing key forms of compositional generalization. Their approach predicts execution subgoals to solve problems incrementally, guided by program execution at each step as well as carefully curated few shot LLM prompts. For text2sql, Eyal et al. (2023) propose an intermediate Query Plan Language (QPL) more amenable to decomposition that SQL. Lately, LLMs (Pourreza and Rafiei, 2024; Liu and Tan, 2023; Wang et al., 2023) — suitably prompted with ICEs and chain-of-thought (CoT) reasoning — perform schema linking, query decomposition, SQL generation, and refinement for text2sql tasks.

These methods rely on static few-shot examples and decomposition of the *test* utterance, which can incur high latency and computational overhead. In contrast, our approach applies decomposition to the labeled *training* instances, creating a larger ICE pool. We then rely on test-time selection to identify

Original Utterance	Code	For sub-tree of the original utterance		
		Generated Utterance (Dependent on original utterance)	Generated Utterance (Independent of original utterance)	Comment
Schedule a team meeting tomorrow from 3 : 00 pm to the end of the day	CreateEvent(ends_at(**EndOfWorkDay**( )))	Schedule event to end at the end of the day	Schedule an event that ends at the **end of the workday**.	Uses function name to add unnatural words to the query
I need to book a meeting in the conference room with Lynne , Howard , Rick and myself for Friday the 10 th at 9 AM please	CreateEvent(AND(starts_at(**nextDayOfMonth**(Today( ) , 10)) , starts_at(NumberAM(9))))	book a meeting for Friday the 10 th at 9 AM	Schedule an event for the 10th of **next month** at 9 AM.	Uses function name to add extra/incorrect info into the query
Put me as busy from 2 pm til 6 pm for lunch tomorrow	do(Let(x0 , **NextTime**(time= NumberPM(2))) , CreateEvent(AND(starts_at($ x0) , ends_at(AND(GE($ x0) , NumberPM(6))) , has_status(ShowAsStatus(Busy)))))	Put me as busy from 2 pm til 6 pm	Schedule me as busy from 2 PM to 6 PM **next time it's 2 PM.**	Uses function name to add extra/incorrect info into the query
Let 's make a lunch meeting with Ryan and Jill at the Starbucks Cafe on 5 th at 4 PM today	CreateEvent(has_subject(lunch))	make a lunch meeting	What is the subject of the event you want to create?	Incorrect interpretation of the function
Let 's make a lunch meeting with Ryan and Jill at the Starbucks Cafe on 5 th at 4 PM today	CreateEvent(starts_at(**nextDayOfMonth(Today( ) , 5)**))	make event on 5 th today	Schedule an event for the **5th of next month**.	Uses function name to add extra/incorrect info into the query

Figure 3: Examples showing how utterances generated by SCUD4ICL conditional on original training utterances are more fluent and natural than utterances generated when the LLM is not prompted to encourage subsumption.

diverse and relevant ICEs (often choosing subtrees of the original ICEs). This approach avoids high latency and computational costs of test utterance decomposition during inference.

ICL issues affecting compositional generalization Given the broad applicability of ICL, numerous studies have explored how factors such as the model scale, token novelty, test query complexity, and demonstration placement influence overall performance. Qiu et al. (2022) show that larger model sizes improve compositional generalization in semantic parsing, but heightens sensitivity to noise in ICEs, a trend also noted by Shi et al. (2023a); Wei et al. (2023) and explained theoretically by Shi et al. (2024). Building on this observation, Wang et al. (2024) introduce a model-specific demonstration selection method tailored to LLM biases. An et al. (2023) investigate the impact of similarity, diversity, and complexity in ICEs, concluding that optimal performance comes from examples that are structurally similar, diverse, and simple. They also emphasize the need for the ICEs to cover the linguistic structures of the test instance.

3 Proposed method: SCUD4ICL

We are given a training set $\mathcal{T} = \{(\mathbf{x}_i, \mathbf{y}_i) : i = 1, \dots, N\}$, comprising N pairs of natural language utterances $\mathbf{x}_i$ and their corresponding programs $\mathbf{y}_i$. Our goal is to use $\mathcal{T}$ to provide an LLM acting as a semantic interpreter) a set S of M in-context examples for any test question $\mathbf{x}_0$. Unlike previous methods, where S contained a subset of only the original examples in $\mathcal{T}$, our method SCUD4ICL seeks more focused sub-programs to reduce interference during in-context learning. We achieve this in two steps: (1) We perform a one-time decomposition of each training instance to create an augmented training pool $\mathcal{T}_{\text{D}}$, and (2) Next, for each test utterance $\mathbf{x}_0$, we select a focused, diverse set of examples from $\mathcal{T}_{\text{D}}$. We describe these steps next.

3.1 Instance decomposition

Given an utterance-program pair $(\mathbf{x}_i, \mathbf{y}_i)$, our goal is to decompose $\mathbf{y}_i$ into meaningful sub-programs $\mathbf{y}_{i,1}, \dots, \mathbf{y}_{i,K_i}$ and associate each sub-program $\mathbf{y}_{i,k}$ with a corresponding utterance $\mathbf{x}_{i,k}$ (which may have to be generated artificially). We assume that $\mathbf{y}_i$ is a semantic parse tree, and the tree structure naturally defines sub-programs corresponding to its subtrees. We show an example of a decomposition in Figure 2.

Sub-utterance generation A baseline method to generate decompositions is to first extract sub-trees rooted at each internal node of the original tree $\mathbf{y}_i$ following the grammar of the program. Then for each subtree $\mathbf{y}_{i,j}$, invoke an LLM to generate utterances for the subtrees, possibly using ICL for that intermediate task. We call this the *"independent utterance decomposition"* method.

A limitation of the above method is that the synthetically generated utterance may not align with the style and language of human-generated utterance, and thus may not be useful demonstration for converting actual test utterances into programs. Our key idea is to view $\mathbf{x}_{i,j}$ as a *sub-utterance* of $\mathbf{x}_i$. We harness LLMs for generating (sub-utterance, sub-tree) pairs conditional on an $\mathbf{x}_i, \mathbf{y}_i$. We call this the *"subsumed utterance decomposition"* method. The LLM is instructed to preserve as much of the original utterance in generating the sub-utterance, and also shown a few (manually created) examples of such decompositions. In Figure 2, notice how each of generated sub-utterances are almost subsumed by the original utterance while being faithful to the given sub-tree. In Figure 3, we contrast the utterance generated by our conditional prompting against independent generations. Note how the ut-

```
In this task, your goal is to decompose
complex event scheduling queries into
simpler, self-contained sub-queries.

Each sub-query should be represented
as a key-value pair within a JSON
object.  The key is the sub-query,
and the value is an object containing
the corresponding Domain-Specific
Language (DSL) code and any further
decompositions.

The primary objective is to ensure that
the decompositions closely mirror the
language used in the original query.
This is to retain the natural language
and idiosyncrasies of the user's input,
which are crucial for understanding the
context and intent of the query.

Each sub-query should be able to stand
on its own, without relying on the
context of the original query.  This
means that the sub-queries should be
clear and unambiguous, even when viewed
independently of the original query.
```

Figure 4: Instruction to LLM for subsumed utterance decomposition in SMCalFlow. These are followed by a few decomposition ICEs. Figure 12 shows a sample.

terances generated independent for each sub-tree appear too verbatim and unnatural compared to our subsumed generation. Figure 4 shows the prompt used for such generation.

3.2 ICE selection from $\mathcal{T}_{\text{D}}$

Let $\mathcal{T}_{\text{D}}$ denote the original dataset $\mathcal{T}$ augmented with generated decomposed pairs $\{(\mathbf{x}_{n,k}, \mathbf{y}_{n,k}) : n = 1, \dots, N; k = 1 \dots K_i\}$ pairs. Given a test question $\mathbf{x}$, we adapt the state of the art diverse decomposition algorithm of Levy et al. (2023) to select examples from $\mathcal{T}_{\text{D}}$. Let M be the budgeted number of ICL examples to select. We collect ICEs one by one. At each step, a new candidate $(\mathbf{x}_{n,k}, \mathbf{y}_{n,k})$ is selected if it satisfies these criteria:

1. utterance or sub-utterance $\mathbf{x}_{n,k}$ covers one or more hitherto uncovered token(s) in the test utterance $\mathbf{x}_0$,
2. a descendant or ancestor node of the root of $\mathbf{y}_{n,k}$ has not already been selected,
3. when the anonymized version of $\mathbf{y}_{n,k}$ (i.e., with all entity names/values replaced by a common token ANON) is not among the ones already seen, and
4. $\mathbf{x}_{n,k}$ is maximally similar to $\mathbf{x}_0$ among candidates that satisfy the above criteria (similarity is measured using BM25).

Algorithm 1 presents our ICE selection strategy. As can be seen, various tweaks to diverse decomposi-

Algorithm 1 SCUD4ICL Example Selection

Require:
$\mathcal{T}_{\text{D}}$: decomposed ICE pool $\{(\mathbf{x}_{n,k}, \mathbf{y}_{n,k}) : n \in [N], k \in [K_n]\}$
M: budget for selected examples
Test query: $\mathbf{x}_0$
Ensure: Diverse and relevant selected examples S
$S \leftarrow \emptyset$ /* Selected examples */
$T_{\text{covered}} \leftarrow \emptyset$ /* Covered test tokens */
$T_{\text{seen}} \leftarrow \emptyset$ /* Seen anonymized templates */
while $|S| < M$ **do**
  $T_{\text{promising}} \leftarrow \emptyset$
  for each token $x \in \mathbf{x}_0$ such that $x \notin T_{\text{covered}}$ **do**
    Identify candidates (n, k) in $\mathcal{T}_{\text{D}}$ satisfying:
      1. $(n, k) \notin S$
      2. No ancestor/descendant of $\mathbf{y}_{n,k}$ is in S
      3. anonymized$(\mathbf{y}_{n,k}) \notin T_{\text{seen}}$
      4. $x \in \mathbf{x}_{n,k}$
    if valid candidates exist **then**
      $T_{\text{promising}} \leftarrow T_{\text{promising}} \cup \{x\}$
  if $T_{\text{promising}} \subseteq T_{\text{covered}}$ **then**
    $T_{\text{covered}} \leftarrow \emptyset$ /* Reset coverage */
    continue
  if $T_{\text{promising}} = \emptyset$ **then**
    break
  $x \leftarrow$ random token from $T_{\text{promising}}$
  $C \leftarrow$ set of candidates in $\mathcal{T}_{\text{D}}$ for token x
  $(n^*, k^*) \leftarrow \arg\max_{(n,k) \in C} \text{BM25}(\mathbf{x}_0, \mathbf{x}_{n,k})$
  $S \leftarrow S \cup \{(n^*, k^*)\}$
  $T_{\text{seen}} \leftarrow T_{\text{seen}} \cup \{\text{anonymized}(\mathbf{y}_{n^*,k^*})\}$
  $T_{\text{covered}} \leftarrow T_{\text{covered}} \cup \{x\}$
return S

tion (Levy et al., 2023) are needed to accommodate candidates corresponding to program fragments that may have structured relations between them, including matching against canonical anonymized structural sketches of ICE sub-programs already accepted into S. Algorithm 1 can be viewed as solving an optimization problem balancing relevance to the test query $\mathbf{x}_0$ and diversity within the selected set S constrained by a budget M using an efficient greedy algorithm. Refer to Appendix B for a more detailed discussion.

4 Experimental Setup

We experiment with two semantic parsing datasets.

SMCalFlow (Andreas et al., 2020) SMCalFlow-CS is a dataset of approximately 25 thousand human-generated utterances about calendar management. To mimic realistic settings of limited labeled data, we created two different training sets, $T5$ and $T10$, as follows. For each test sample, we selected 5 (respectively, 10) closest samples on utterance-level cosine similarity. We retain those of depth greater than three. This yielded 71 unique instances in $T5$, and 116 in $T10$. After decomposition with our method we obtained 296 and 473 non-leaf decompositions which we treat as the aug-

LLM	ICL type	SMCalFlow				GeoQuery			
		EN	HI	FR	RU	length	i.i.d	template	tmcd
GPT3.5	CoverLS	42.4	43.4	40.1	48.7	50.9	77.2	86.5	70.9
	SCUD4ICL	**45.8**	**50.2**	**46.0**	**49.5**	**59.2**	**81.9**	**87.7**	**82.0**
GPT4o	CoverLS	54.1	49.6	53.5	54.5	73.6	86.3	91.6	80.7
	SCUD4ICL	**54.2**	**51.8**	**54.3**	**55.8**	**81.0**	**86.9**	89.8	**85.4**
Mistral:7b	CoverLS	45.8	32.7	43.5	36.5	32.3	58.9	62.3	49.4
	SCUD4ICL	**46.6**	**37.2**	**46.0**	**39.8**	**50.6**	**67.7**	**74.3**	**73.9**
Llama3:8b	CoverLS	48.2	28.5	43.5	39.3	35.6	55.8	63.4	50.9
	SCUD4ICL	46.4	**34.2**	**43.7**	**40.2**	**44.4**	52.9	58.9	48.4

Table 1: Execution accuracy on two datasets: (1) SMCalFlow with training split $\mathcal{T}$ = T5 with programs in four different languages En,Hi,Fr,Ru. and (2) GeoQuery with training split $\mathcal{T}$ = D3 and various test splits. All results are average of three runs and with $M = 5$. Observe that SCUD4ICL provides much higher accuracy than CoverLS in most cases, and the gains are higher for smaller LMs.

LLM	ICL type	EN		FR	
		random	full	random	full
M=5					
Mistral:7b	CoverLS	25.0	22.5	17.3	18.2
	SCUD4ICL	**27.3**	**22.8**	**19.7**	**19.6**
Llama3:8b	CoverLS	17.3	14.9	11.0	11.3
	SCUD4ICL	**18.7**	**16.5**	**11.3**	**12.1**
M=10					
Mistral:7b	CoverLS	29.7	24.0	23.0	21.1
	SCUD4ICL	**32.0**	**26.4**	**25.3**	**22.9**
Llama3:8b	CoverLS	17.3	16.1	13.7	13.0
	SCUD4ICL	**23.7**	**20.8**	**15.3**	**14.6**

Table 2: Exact match accuracy on MTOP for EN and FR for $M = 5$ and $M = 10$. Results on random set are averaged over three runs.

LLM	ICL type	EN	HI
GPT3.5	DPP	19.3	24.6
	TOPK	24.9	25.5
	CoverLS	42.4	43.2
	SCUD4ICL	**45.7**	**50.5**
Mistral:7b	DPP	22.9	23.4
	TOPK	25.3	24.0
	CoverLS	45.4	32.1
	SCUD4ICL	**46.0**	**36.7**
Llama3:8b	DPP	27.0	22.5
	TOPK	27.9	22.0
	CoverLS	**48.0**	27.6
	SCUD4ICL	45.3	**34.1**

Table 3: SCUD4ICL vs. other baselines on SMCalFlow (training split: $\mathcal{T}$ = T5, $M = 5$).

mented training set $\mathcal{T}_{\text{D}}$. The default train pool is $T5$. In these experiments we were concerned if the recent LLMs were already trained or fine-tuned on these datasets (data contamination). To partially mitigate this concern, we created three variants of the data, where English names in clauses are replaced by their counterpart in three other languages: Hindi, French, and Russian. The utterance $\mathbf{x}_i$ stays in English, only in $\mathbf{y}_i$ the names of clauses are changed using a mapping as shown in Figure 8, 9 and 10 in the Appendix. We will soon see that changing clause names impacts performance, particularly in recent smaller LLMs.

GeoQuery (Zelle and Mooney, 1996) is a dataset of 880 user utterances seeking geographical information, such as locations of rivers and cities. ***Test splits:*** For test data we use the *iid* (standard) and compositional splits created by Shaw et al. (2021) as follows: (1) *template* split where programs output templates instead of grounded values for arguments (Finegan-Dollak et al., 2018); (2) *TMCD* split, with divergent distributions of compounds in training and test sets (Keysers et al., 2019); and (3) *length* split, where test sequences are longer than training ones. As in prior work, we average results over three TMCD and template splits to reduce variance from small dataset size.
Train splits: Out of the available labeled dataset, we considered two subsets for defining the candidate labeled pool $\mathcal{T}$: (1) D3, comprising of trees of depth ≥ 3 (default), and (2) D4, comprising of trees of depth ≥ 4. For the *length* split, since the test splits includes all the larger trees, the train split had only 8 trees left of depth 4. So we do not consider this train-test split. Across the other splits, the size of $D3$ is roughly four times the size of $D4$. Exact counts appear in Table 4.

MTOP (Li et al., 2021) is a task-oriented dialogue dataset that maps user commands to complex, nested queries spanning 11 domains. We utilize the English and French subsets in our experiments. The original training sets include 15,667 English and 11,814 French utterances; we retain instances with query depth of at least 5, reducing them to 416 English and 211 French examples. For controlled cross-lingual evaluation, we further filter the data to include only instances with corresponding queries in the other language, yielding a parallel training set of 108 examples. The test set comprises 1,713 parallel examples per language.

Models and Prompts We evaluate our method and baselines on the following LLMs: GPT 3.5[3], GPT-4o[4], Mistral[5]-7b v0.3, and LLama3–8b (Dubey et al., 2024). The prompt used for decomposition is shown in Figure 4. We use GPT4_0125 to get the decompositions. The prompt used for generating semantic parses is shown in Figure 6.

Evaluation Metric We evaluate our approach using execution accuracy (EX), which measures the correctness of the outputs produced by the predicted programs as well as exact match (EM) accuracy depending on the dataset. Following (Bogin et al., 2024), we compute EX by comparing the execution results of predicted and gold programs. For GeoQuery, we compare answers returned by generated programs to those generated by gold programs, while for SMCalFlow, we compare the state (i.e., calendar events) of the environments after execution. In case of MTOP, we measure the exact match accuracy.

Baseline Our method of augmenting the training set with decompositions of the original question is largely orthogonal to the algorithm used for selecting in-context examples for a test question $\mathbf{x}$. Therefore, as a baseline, we choose the state-of-the-art algorithm CoverLS (Levy et al., 2023) for selecting examples from the original training set. We used the official code released by the authors[6].

5 Results

We first present the performance of the proposed algorithm relative to various baselines and then discuss various ablations to dissect the reason for the gains.

SCUD4ICL vs. Baselines In Tables 1, 2, and 3, we compare accuracy of SCUD4ICL with the baseline CoverLS across various datasets and splits discussed earlier. Results with additional splits are presented in the Appendix 11. Based on the results, we make the following observations.

[3]https://platform.openai.com/docs/models/gpt-3-5-turbo

[4]https://openai.com/index/hello-gpt-4o/

[5]https://mistral.ai/news/announcing-mistral-7b/

[6]https://github.com/itayle/diverse-demonstrations

(1) First, across both datasets and varying test-train splits, SCUD4ICL provides much higher accuracy than CoverLS and significantly outperforms DPP and TOPK (Levy et al., 2023). The main reason for the gains over CoverLS is the augmentation of the training pool with decomposed training instances since the algorithm used for selecting the M instances are largely similar.
(2) As expected, just by changing the language in the names of clauses, accuracy varies across all LLMs. For a low resource language such as Hindi (Hi), SCUD4ICL provides much higher gains than on English. For example, with GPT3.5 and Llama3:8b we observe more than 6% absolute jump in accuracy on SMCalFlow-Hi.
(3) SCUD4ICL provides better generalization when train-test splits differ in length as seen from Geoquery's *length* test split. Even for GPT4o we observe a jump in accuracy from 73.6 to 81, and for Mistral:7b accuracy jumps from 32.3% to 50.6%.
(4) SCUD4ICL also generalizes better when there is discrepancy in the template of the test and train trees as seen in Geoquery's *tmcd* test split. For GPT3.5 accuracy jumps from 70.9% to 82%, and for Mistral:7b the jump is from 49.4% to 73.9%.

Decomposition variants We analyze whether the gains are just because of augmenting the train pool $\mathcal{T}$ with decomposed sub-trees, or whether the quality of the sub-utterance was the key reason. In Table 5 we demonstrate the impact of our method of generating subsumed sub-utterances by comparing with independently generated sub-utterances as described in Section 3.1. We observe that with independent sub-utterances, the performance is similar to the original un-augmented data. Only with our subsumed sub-utterance, do we get the accuracy gains seen above. The examples in Figure 3 illustrate that independent utterances often do not appear natural enough, and they possibly fail to match test utterances. Another question is whether sub-utterances generated by a lower capacity open LLM such as Mixtral:8x22b (Mistral AI, 2024) compare with those from GPT4_0125. In Table 7 we show accuracy with SMCalFlow-Hi. Observe that even with Mixtral:8x22b-generated sub-utterances, SCUD4ICL provides adequate gains over the baseline, although the gains are greater with GPT4o.

Impact of Fragment Selection To examine the impact of fragment selection in in-context examples, SCUD4ICL-selected fragments were compared with the corresponding full ICE examples,

LLM	ICL type	SMCalflow								Geoquery							
		M=5 Languages				M=10 Languages				M=5 Test Splits				M=10 Test Splits			
		EN	HI	FR	RU	EN	HI	FR	RU	length	iid	template	tmcd	length	iid	template	tmcd
Train Split		$\mathcal{T}$ = T5								$\mathcal{T}$ = D4							
No. of instances		71/296								2/0	54/176	35/120	34/107	2/0	54/176	35/120	34/107
GPT3.5	**CoverLS**	42.4	43.4	40.1	48.7	42.6	50.1	41.4	47.1	-	64.4	69.9	63.4	-	71.5	73.7	67.3
	SCUD4ICL	**45.8**	**50.2**	**46.0**	**49.5**	**50.8**	**55.8**	**50.9**	**53.3**	-	**69.4**	**71.3**	**66.0**	-	**78.6**	**81.3**	**68.5**
GPT4o	**CoverLS**	54.1	49.6	53.5	54.5	**67.6**	**64.6**	**67.2**	**67.0**	-	74.3	**85.5**	**72.5**	-	81.0	**87.0**	75.9
	SCUD4ICL	**54.2**	**51.8**	**54.3**	**55.8**	65.4	64.0	65.9	66.4	-	**76.8**	81.4	71.9	-	**84.0**	85.9	**77.1**
Mistral:7b	**CoverLS**	45.8	32.7	43.5	36.5	**53.7**	35.3	49.0	43.6	-	45.7	**51.5**	**45.9**	-	54.2	54.8	47.4
	SCUD4ICL	**46.6**	**37.2**	**46.0**	**39.8**	52.3	**40.9**	**52.8**	**45.7**	-	**51.8**	51.1	43.8	-	**58.2**	**59.1**	**47.8**
Llama3:8b	**CoverLS**	**48.2**	28.5	43.5	39.3	**57.9**	33.8	52.1	46.8	-	36.7	41.9	39.4	-	51.2	48.2	45.9
	SCUD4ICL	46.4	**34.2**	**43.7**	**40.2**	57.3	**39.6**	**52.7**	**50.2**	-	**50.0**	**47.6**	**43.1**	-	**61.9**	**61.6**	**51.8**
Train Split		$\mathcal{T}$ = T10								$\mathcal{T}$ = D3							
No. of instances		116/473								64/202	208/701	147/576	131/646	64/202	208/701	147/576	131/646
GPT3.5	**CoverLS**	44.9	49.3	41.3	48.7	44.1	53.1	44.5	51.5	50.9	77.2	86.5	70.9	54.5	80.5	**88.3**	73.4
	SCUD4ICL	**50.7**	**54.8**	**52.5**	**55.0**	**54.3**	**58.6**	**53.8**	**58.1**	**59.2**	**81.9**	**87.7**	**82.0**	**64.4**	**84.6**	88.1	**86.3**
GPT4o	**CoverLS**	56.9	**56.6**	**58.7**	**62.0**	**70.3**	**69.3**	67.6	**70.9**	73.6	86.3	**91.6**	80.7	81.2	88.4	**93.8**	84.1
	SCUD4ICL	**58.1**	56.4	58.0	61.1	69.0	67.0	**67.7**	69.9	**81.0**	**86.9**	89.8	**85.4**	**82.4**	**89.0**	91.5	**90.8**
Mistral:7b	**CoverLS**	**50.2**	38.4	**48.5**	**42.0**	55.1	41.7	52.4	48.4	32.3	58.9	62.3	49.4	37.2	62.5	69.5	56.6
	SCUD4ICL	48.1	**39.2**	47.2	41.2	**55.9**	**44.9**	**54.3**	**49.9**	**50.6**	**67.7**	**74.3**	**73.9**	**57.9**	**74.5**	**82.3**	**77.3**
Llama3:8b	**CoverLS**	**51.2**	35.3	46.4	**44.3**	**59.1**	37.1	52.8	52.1	35.6	**55.8**	**63.4**	**50.9**	40.1	**63.8**	**70.1**	55.6
	SCUD4ICL	50.5	**37.8**	**49.0**	44.2	58.7	**44.6**	**56.4**	**53.2**	**44.4**	52.9	58.9	48.4	**55.5**	61.0	62.0	**56.3**

Table 4: Execution accuracy on SMCalflow across different code languages and GeoQuery across different types of test splits while increasing M from 5 to 10, and increasing size of training pool $\mathcal{T}$ (top and bottom). Please see Sec 4 for reasons for some missing numbers. The number of instances denote size of $\mathcal{T}$ and $\mathcal{T}_{\mathrm{D}}$ separated by '/'. The broad trend is that accuracy gains of SCUD4ICL is higher for larger M and smaller $\mathcal{T}$.

LLM	**ICL type**	EN	HI
GPT3.5	CoverLS	42.4	43.4
	Independent	43.4	45.0
	SCUD4ICL	**45.8**	**50.2**
GPT4o	CoverLS	54.1	49.6
	Independent	53.3	48.4
	SCUD4ICL	**54.2**	**51.8**
Mistral:7b	CoverLS	45.8	32.7
	Independent	46.8	35.8
	SCUD4ICL	46.6	**37.2**
Llama3:8b	CoverLS	48.2	28.5
	Independent	47.5	29.9
	SCUD4ICL	46.4	**34.2**

Table 5: Comparison of Independent Vs Subsumed sub-utterance generation for $M = 5$, training split $\mathcal{T}$ = T5(SMCalFlow) on execution accuracy. Observe how Independent provides almost no gains over the baseline CoverLS in spite of including exactly the same set of sub-trees in $\mathcal{T}_{\mathrm{D}}$. Thus, SCUD4ICL's method of generating sub-utterances subsumed by the original utterance is a key reason for its gains.

referred to as WholeExamples. Both approaches were evaluated using the same set of examples to ensure a controlled comparison.

The results shown in Table 8 indicate that SCUD4ICL consistently achieves superior or comparable performance compared to WholeExamples across all evaluated models and languages. This demonstrates that the removal of irrelevant parts in ICEs reduces distractions and contributes to improved predictive performance.

Accuracy gains for different test tree sizes We show a breakdown of the accuracy gains across depth of the test trees in Table 6. For a mainstream language like English depth 2 trees may be easy enough, and we do not see much gains with SCUD4ICL. For low resource language like Hindi, biggest gains are obtained from shorter trees (depth=2) that are more subject to interference from irrelevant clauses in the decomposition.

Increasing M and training pool $\mathcal{T}$ In Table 4 we show accuracy with M, the size of the in-context set, increasing from 5 to 10, and two different training sizes. A rough trend to observe is that for larger M, SCUD4ICL's gains over baseline increases. Consider, for instance Geoquery's *iid* split on the D3 train set. Across all LLMs, SCUD4ICL provides higher gains with $M = 10$ than with $M = 5$ on this train-test split. Such a trend is explained by the fact that baseline is more likely to include irrelevant clauses for large M, and our decomposition is able to eliminate them.

When the training pool $\mathcal{T}$ size increases with a fixed M, the baseline is likely to find increasingly relevant instances for ICL. Thus, we expect the impact of decomposition to be higher when the training pool is smaller. In Table 4 we observe that relative gains are higher in the top-half compared to the bottom half. We zoom in further on

LLM	ICL type	EN				HI				FR				RU			
		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
GPT3.5	CoverLS	50.0	52.1	36.6	53.7	50.0	48.9	40.3	44.4	50.0	45.7	37.4	31.5	50.0	54.3	45.8	44.4
	SCUD4ICL	50.0	51.2	**42.9**	42.6	50.0	**57.5**	**46.2**	**46.3**	50.0	**54.4**	**41.9**	**33.3**	50.0	**55.3**	**46.0**	**55.6**
GPT4o	CoverLS	50.0	63.9	48.9	48.1	50.0	58.5	45.0	42.6	50.0	60.4	49.0	68.5	50.0	63.0	49.6	57.4
	SCUD4ICL	50.0	62.9	**49.8**	44.4	50.0	**64.2**	**45.5**	37.0	50.0	**61.0**	**50.5**	55.6	50.0	62.9	**52.1**	53.7
Mistral:7b	CoverLS	0.0	47.5	45.1	44.4	0.0	33.9	32.7	22.2	0.0	46.3	42.3	40.7	0.0	38.4	35.5	37.0
	SCUD4ICL	0.0	44.0	**48.4**	44.4	**50.0**	**39.0**	**37.3**	11.1	**50.0**	**50.1**	**44.0**	38.9	0.0	**40.3**	**40.0**	31.5
Llama3:8b	CoverLS	0.0	47.6	48.5	53.7	0.0	32.9	27.4	1.9	0.0	44.3	43.4	40.7	50.0	43.1	37.6	29.6
	SCUD4ICL	0.0	45.4	46.9	53.7	0.0	**39.3**	**32.4**	**16.7**	**50.0**	44.3	**43.7**	35.2	33.3	**43.5**	**38.3**	**42.6**

Table 6: Execution accuracy broken down by depth of the test tree. We consider depth values 1, 2, 3, 4. Almost 30% of the test trees are of depth 2, and 55% of depth 3. For a mainstream language like EN, depth 2 trees seem easy enough for the LLM, with most gains from SCUD4ICL on trees of depth 3. For an unfamiliar language like HI, depth 2 trees also benefit substantially from SCUD4ICL, since interference is likely highest for small trees.

LLM	ICL type		
	CoverLS	Mixtral8x22b	GPT4
GPT3.5	43.4	49.9	**50.2**
GPT4o	49.6	51.1	**51.8**
Mistral:7b	32.7	35.1	**37.2**
Llama3:8b	28.5	**34.5**	34.2

Table 7: Comparison of GPT4_0125 vs Mixtral8x22b decomposition. Even with decompositions from a smaller LLM, SCUD4ICL improves over baseline.

LLM	ICL type	EN	HI
GPT3.5	SCUD4ICL	**45.8**	**50.2**
	WholeExamples	42.5	44.7
GPT4o	SCUD4ICL	**54.2**	**51.8**
	WholeExamples	50.6	48.4
Mistral:7b	SCUD4ICL	**46.6**	**37.2**
	WholeExamples	45.7	34.5
Llama3:8b	SCUD4ICL	46.4	**34.2**
	WholeExamples	**46.8**	27.3

Table 8: Comparison of SCUD4ICL and WholeExamples on SMCalFlow (training split: $\mathcal{T}$ = T5, $M = 5$).

SMCalFlow-Hi for $M = 5$ in Figure 5. Observe the increased relative gains $\mathcal{T} = T5$ with 71 instances compared to $\mathcal{T} = T10$ with 116 instances.

In summary, our experiments show that while SCUD4ICL provides overall gains over baseline, it is particularly useful when the code language is less familiar, there is mismatch in the train and test distribution in terms of code length and template, IC budget is large, and the training pool is small.

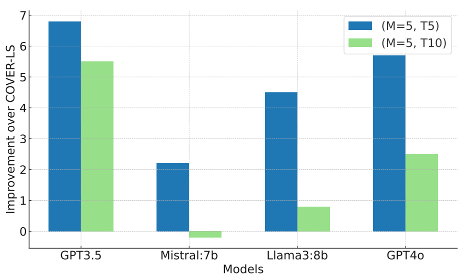

Figure 5: Accuracy gains of SCUD4ICL over baseline for SMCalFlow-Hi version for two different training pool sizes pointing to higher gains for a smaller pool.

6 Conclusion

We introduced SCUD4ICL, a semantic interpreter that incorporates a new paradigm of fragmenting structured programs from the pool of available ICEs, and instructing an LLM to translate these program fragments back to natural language utterances. This forms a larger ICE pool including decomposed (utterance, program) pairs. Test instance-guided diverse ICE selection from this enlarged pool improves semantic interpretation accuracy for a number of benchmarks.

7 Limitations

Although use of commercial LLMs as network services is widespread in this nature of research, it severely reduces reproducibility. As one example, we do not understand how the LLM implements utterance subsumption. Should the LLM change significantly owing to instruction tuning, our results may change drastically. Further experiments with prompt/prefix tuning, and/or setting up a smaller in-house LLM and adapting it to our task, would be of future interest.

References

Shengnan An, Zeqi Lin, Qiang Fu, Bei Chen, Nanning Zheng, Jian-Guang Lou, and Dongmei Zhang. 2023. How do in-context examples affect compositional generalization? In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11027–11052, Toronto, Canada. Association for Computational Linguistics.

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.

Ben Bogin, Shivanshu Gupta, Peter Clark, and Ashish Sabharwal. 2024. Leveraging code to improve in-context learning for semantic parsing. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4971–5012, Mexico City, Mexico. Association for Computational Linguistics.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Ma teusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *ArXiv*, abs/2005.14165.

Jiuhai Chen, Lichang Chen, Chen Zhu, and Tianyi Zhou. 2023. How many demonstrations do you need for in-context learning? In *Conference on Empirical Methods in Natural Language Processing*.

Abhimanyu Dubey et al. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Ben Eyal, Amir Bachar, Ophir Haroche, Moran Mahabi, and Michael Elhadad. 2023. Semantic decomposition of question and sql for text-to-sql parsing. *arXiv preprint arXiv:2310.13575*.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia. Association for Computational Linguistics.

Yanhe Fu, Yanan Cao, Qingyue Wang, and Yi Liu. 2024. TISE: A tripartite in-context selection method for event argument extraction. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*.

Shivanshu Gupta, Matt Gardner, and Sameer Singh. 2023. Coverage-based example selection for in-context learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 13924–13950, Singapore. Association for Computational Linguistics.

SU Hongjin, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, et al. 2022. Selective annotation makes language models better few-shot learners. In *The Eleventh International Conference on Learning Representations*.

Cheng-Yu Hsieh, Sibei Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander J. Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. 2023. Tool documentation enables zero-shot tool-usage with large language models. *ArXiv*, abs/2308.00675.

Xiang Huang, Sitao Cheng, Yiheng Shu, Yuheng Bao, and Yuzhong Qu. 2023. Question decomposition tree for answering complex questions over knowledge bases. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12924–12932.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, et al. 2019. Measuring compositional generalization: A comprehensive method on realistic data. *arXiv preprint arXiv:1912.09713*.

Alex Kulesza and Ben Taskar. 2011. Learning determinantal point processes. *Learning*, 7:1–2011.

Itay Levy, Ben Bogin, and Jonathan Berant. 2023. Diverse demonstrations improve in-context compositional generalization. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 1401–1422. Association for Computational Linguistics.

Haoran Li, Abhinav Arora, Shuohui Chen, Anchit Gupta, Sonal Gupta, and Yashar Mehdad. 2021. MTOP: A comprehensive multilingual task-oriented semantic parsing benchmark. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2950–2962, Online. Association for Computational Linguistics.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023a. Lost in the middle: How language models use long contexts. *Preprint*, arXiv:2307.03172.

Xiping Liu and Zhao Tan. 2023. Divide and prompt: Chain of thought prompting for text-to-sql. *arXiv preprint arXiv:2304.11556*.

Ye Liu, Semih Yavuz, Rui Meng, Dragomir Radev, Caiming Xiong, and Yingbo Zhou. 2023b. Hpe: Answering complex questions over text by hybrid question parsing and execution. *arXiv preprint arXiv:2305.07789*.

Sheng Lu, Irina Bigoulaeva, Rachneet Sachdeva, Harish Tayyar Madabushi, and Iryna Gurevych. 2023. Are emergent abilities in large language models just in-context learning? *ArXiv*, abs/2309.01809.

Mistral AI. 2024. Mixtral 8x22b: A pretrained generative sparse mixture of experts. https://mistral.ai/en/news/mixtral-8x22b. Mixtral 8x22B is our latest open model. It sets a new standard for performance and efficiency within the AI community. It is a sparse Mixture-of-Experts (SMoE) model that uses only 39B active parameters out of 141B, offering unparalleled cost efficiency for its size.

Mohammadreza Pourreza and Davood Rafiei. 2024. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Tianze Shi, Jonathan Herzig, Emily Pitler, Fei Sha, and Kristina Toutanova. 2022. Evaluating the impact of model scale for compositional generalization in semantic parsing. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9157–9179, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2021. Learning to retrieve prompts for in-context learning. *arXiv preprint arXiv:2112.08633*.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2021. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? 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 922–938, Online. Association for Computational Linguistics.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed Chi, Nathanael Schärli, and Denny Zhou. 2023a. Large language models can be easily distracted by irrelevant context. *Preprint*, arXiv:2302.00093.

Kensen Shi, Joey Hong, Yinlin Deng, Pengcheng Yin, Manzil Zaheer, and Charles Sutton. 2023b. Exedec: Execution decomposition for compositional generalization in neural program synthesis. *arXiv preprint arXiv:2307.13883*.

Zhenmei Shi, Junyi Wei, Zhuoyan Xu, and Yingyu Liang. 2024. Why larger language models do in-context learning differently? *Preprint*, arXiv:2405.19592.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2023. Mac-sql: Multi-agent collaboration for text-to-sql. *arXiv preprint arXiv:2312.11242*.

Huazheng Wang, Jinming Wu, Haifeng Sun, Zixuan Xia, Daixuan Cheng, Jingyu Wang, Qi Qi, and Jianxin Liao. 2024. MDR: Model-specific demonstration retrieval at inference time for in-context learning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4189–4204, Mexico City, Mexico. Association for Computational Linguistics.

Jerry Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, and Tengyu Ma. 2023. Larger language models do in-context learning differently. *Preprint*, arXiv:2303.03846.

Weixiang Yan, Haitian Liu, Yunkun Wang, Yunzhe Li, Qian Chen, Wen Wang, Tingyu Lin, Weishan Zhao, Li Zhu, Shuiguang Deng, and Hari Sundaram. 2023. CodeScope: An execution-based multilingual multitask multidimensional benchmark for evaluating llms on code understanding and generation. In *Annual Meeting of the Association for Computational Linguistics*.

Jiacheng Ye, Zhiyong Wu, Jiangtao Feng, Tao Yu, and Lingpeng Kong. 2023. Compositional exemplars for in-context learning. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Junyi Ye, Mengnan Du, and Guiling Wang. 2024. DataFrame QA: A universal LLM framework on DataFrame question answering without data exposure. *ArXiv*, abs/2401.15463.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI'96, page 1050–1055. AAAI Press.

Diverse In-Context Example Selection After Decomposing Programs and Aligned Utterances Improves Semantic Parsing (Appendix)

A Additional Experimental Results

A.1 Exact Match Accuracy

LLM	ICL Type	EN	HI	FR	RU
GPT3.5	CoverLS	18.0	17.0	14.4	17.6
	SCUD4ICL	**21.1**	**19.5**	**18.3**	**19.6**
GPT4o	CoverLS	**18.0**	12.3	**14.9**	16.8
	SCUD4ICL	17.5	**12.9**	14.2	**16.8**
Mistral:7b	CoverLS	**13.8**	**7.7**	**11.7**	9.5
	SCUD4ICL	13.3	6.9	11.3	**9.7**
Llama3:8b	CoverLS	**13.0**	5.8	**10.3**	8.5
	SCUD4ICL	11.6	**7.7**	10.1	**10.7**

Table 9: Exact Match accuracy for Table 1, training split $\mathcal{T}$ = T5 (SMCalFlow)

Table 9 compares SCUD4ICL and CoverLS in terms of exact match accuracy. While SCUD4ICL consistently outperformed CoverLS in execution accuracy (Table 1 in the paper), its performance in terms of exact match accuracy is mixed. SCUD4ICL excels on older models like GPT3.5 and in low-resource languages (e.g., Hindi and Russian), but the lower scores in other cases can be attributed to its approach of assembling code using information in sub-fragment examples. This strategy often produces semantically correct outputs that differ syntactically from the gold standard, leading to lower exact match scores despite maintaining functional correctness.

For example, the query "Hi, can you reschedule me a meeting with Ruth on Monday?" can be translated as either "CreateEvent(AND(starts_at(NextDOW(MONDAY)), with_attendee(Ruth)))" or "CreateEvent(AND(with_attendee(Ruth), starts_at(NextDOW(MONDAY))))". If the former is the gold annotation, the latter is still correct in terms of execution accuracy but fails on exact match.

A.2 Variability in Execution Accuracy Results

The reported results account for randomness, with all values representing the average of three runs. Standard deviations for Table 1 are provided in Table 10.

A.3 Evaluation with Additional Data Splits

Table 11 presents the results comparing CoverLS and SCUD4ICL across the random training sub-

LLM	ICL type	EN	HI	FR	RU
GPT3.5	CoverLS	0.15	0.25	0.75	1.0
	SCUD4ICL	0.30	0.29	0.59	0.46
GPT4o	CoverLS	2.24	1.42	0.38	0.76
	SCUD4ICL	0.46	0.25	0.17	0.84
Mistral:7b	CoverLS	1.08	1.10	1.16	0.75
	SCUD4ICL	0.61	0.55	0.68	0.47
Llama3:8b	CoverLS	0.29	0.86	0.93	0.31
	SCUD4ICL	0.96	0.17	1.01	0.83

Table 10: Standard deviation for Table 1, training split $\mathcal{T}$ = T5 (SMCalFlow)

splits of SMCalFlow (as well as the complete training split for GeoQuery), evaluated using different LLMs and languages. These results indicate that SCUD4ICL consistently outperforms or matches CoverLS leveraging its ability to identify highly relevant sub-fragments, even while using fewer tokens. Note: Random split 1 contains 100 instances and 825 decompositions, whereas random split 2 contains 454 instances and 3,758 decompositions.

A.4 Evaluation of Decomposed Examples in SMCalFlow

An evaluation of the generated sub-utterances was conducted as part of the manual assessment of the SMCalFlow $\mathcal{T}$ = T5 training split. The results are as follows:

- Original questions: 71
- Total decompositions generated: 296
- Incorrectly generated decompositions: 30

This corresponds to a decomposition error rate of 10% of the overall decompositions, with some errors being relatively minor. Despite these errors, experiments demonstrate that the use of decomposed examples leads to improved overall accuracy.

B SCUD4ICL - Optimization Perspective

SCUD4ICL selects examples for in-context learning by solving an optimization problem that balances relevance the test query $\mathbf{x}_0$ and diversity within the selected set S, constrained by a budget M. Our ICE selection follows a greedy approach but can be viewed as an instantiation of the below optimization problem:

$$\max_{S:|S|\leq M} \left[\sum_{(n,k)\in S} \Big(\alpha \cdot R(\mathbf{x}_{n,k}, \mathbf{x}_0) + \beta \cdot D(\mathbf{x}_{n,k}, S) \Big) \right]$$

Here, relevance $R(\mathbf{x}_{n,k}, \mathbf{x}_0)$ measures the similarity between $\mathbf{x}_{n,k}$ and $\mathbf{x}_0$ and diversity

LLM	**ICL type**	**SMCalFlow**				**GeoQuery**			
		Split 1		Split 2		Length	i.i.d	Template	TMCD
		EN	HI	EN	HI	EN			
GPT3.5	CoverLS	43.7	48.8	48.3	56.4	66.4	86.1	80.7	73.0
	SCUD4ICL	**47.4**	**50.8**	**53.0**	**57.7**	**77.0**	**87.1**	**86.7**	**80.0**
Mistral:7b	CoverLS	49.1	36.2	**55.4**	39.2	46.1	71.4	64.8	61.2
	SCUD4ICL	**51.2**	**36.4**	53.3	**42.3**	**64.8**	**74.6**	**70.8**	**65.5**
Llama3:8b	CoverLS	50.6	30.5	**55.4**	36.7	45.2	**63.9**	64.8	56.7
	SCUD4ICL	**51.5**	**34.0**	55.3	**38.4**	**58.8**	62.5	**71.1**	**58.5**

Table 11: Execution accuracy on random training splits of SMCalFlow and full training split of GeoQuery ($M = 5$).

$D(\mathbf{x}_{n,k}, S)$ is defined as the *negative minimal similarity* with other members of S:

$$D(\mathbf{x}_{n,k}, S) = -\min_{(m,l)\in S\setminus\{(n,k)\}} \text{sim}(\mathbf{x}_{n,k}, \mathbf{x}_{m,l}),$$

where $\text{sim}(\mathbf{x}_{n,k}, \mathbf{x}_{m,l})$ measures structural and semantic similarity.

Specific Design Choices in SCUD4ICL

- *Relevance (R)* is computed using BM25 to assess semantic alignment with $\mathbf{x}_0$.
- *Diversity (D)* penalizes similarity to examples already selected, disqualifying candidates with overlaps of their anonymized versions or ancestor-descendant relationships.
- *Token-based Filtering* enables selection of candidates $x_{n,k}$ that contain tokens from x_0 not yet covered, improving efficiency.

To summarize, SCUD4ICL integrates ideas from Diverse Demonstrations (Levy et al., 2023) and DPP-based selection (Ye et al., 2023), while applying structural constraints (e.g., template matching and hierarchical relationships) to improve selection from a decomposed example pool.

C Prompts

In Figures 6 and 7 we sketch the prompt we used to obtain semantic interpretation in the SMCalFlow dataset on two code languages En and Hi respectively.

```
Given the following data structures and functions:

        FindTeamOf  # given a person name or ID, ...
        has_subject  # given a string, returns an ...
        starts_at    # given a datetime clause, ...
        CreateEvent  # given multiple event clauses ...
        ...
        ...


Your task is to write DSL code for the given question.

Note:
1.  Do not use any external libraries/functions.
2.  Strictly adhere to the provided operators.
```

Figure 6: Instruction to LLM for EN code generation (SMCalFlow).

```
Given the following data structures and functions:

        DalKhojen  # given a person name or ID, ...
        VishayHai  # given a string, returns an ...
        SePrarambh  # given a datetime clause, ...
        KaryakramBanao  # given multiple event ...
        ...
        ...


Your task is to write DSL code for the given question.

Note:
1.  Do not use any external libraries/functions.
2.  Strictly adhere to the provided operators.
```

Figure 7: Instruction to LLM for HI code generation (SMCalFlow).

```
FindTeamOf  # given a person name or id, returns a pseudo-person representing the team of that
      person
FindReports  # given a person name or id, returns a pseudo-person representing the reports of that
      person
FindManager  # given a person name or id, returns the manager of that person

with_attendee  # given a person name or id, returns a clause to match or create an event with that
      person as an attendee
avoid_attendee  # given a person name or id, returns an event clause to avoid that attendee when
      creating an event
has_subject  # given a string, returns an event to match or create an event with that subject
at_location  # given a string, returns an event clause to match or create an event at that location
starts_at  # given a datetime clause, returns an event clause to match or create an event starting
      at that time
ends_at  # given a datetime clause, returns an event clause to match or create an event ending at
      that time
has_duration  # given a time unit value, returns an event clause to match or create an event with
      that duration
has_status  # given a ShowAsStatus value, returns an event clause to match or create an event with
      that status

# the following operators return datetime clauses and accept no arguments
Afternoon
Breakfast
Brunch
Dinner
Early
EndOfWorkDay
Evening
FullMonthofMonth
FullYearofYear
LastWeekNew
Late
LateAfternoon
LateMorning
Lunch
Morning
NextMonth
NextWeekend
NextWeekList
NextYear
Night
Noon
Now
SeasonFall
SeasonSpring
SeasonSummer
SeasonWinter
ThisWeek
ThisWeekend
Today
Tomorrow
Yesterday

# general date time clauses
DateTime  # given either a datetime clause representing a date and/or a time operator representing a
       time, returns a datetime clause
Date  # given a date or dayofweek, returns a date
DayOfWeek  # given a day of week string, returns a time clause
NextDOW  # given a day of week string, returns a time clause for the next occurrence of that day of
      week
MD  # given a month and day as arguments, returns a date clause
MDY  # given a month, day, and year as arguments, returns a date clause

# given a value, the following operators return datetime clauses according to the given value
toMonth
toFourDigitYear
HourMinuteAm
HourMinutePm
NumberAM
NumberPM

# given a datetime clause, the following operators modify the clause and return a datetime clause
      according to the modification
OnDateAfterTime
OnDateBeforeTime
AroundDateTime

# given either a number or the operators Acouple/Afew, all the following operators return time unit
      values according to the given unit
toDays
toHours
toMinutes

# these operators can be used to create time unit values instead of using integer values
Acouple
Afew

ShowAsStatus  # enumeration of possible event statuses (Busy, OutOfOffice)

AND  # combines multiple event clauses together
```

Figure 8: Exhaustive list of operators for EN code generation (SMCalFlow).

```
DalKhojen  # given a person name or id, returns a pseudo-person representing the team of that person
ReportDhoondho  # given a person name or id, returns a pseudo-person representing the reports of that
      person
PrabandhakKhojen  # given a person name or id, returns the manager of that person

InSahbhagiyonKeSaath  # given a person name or id, returns a clause to match or create an event with
      that person as an attendee
InSahbhagiyonKeBina  # given a person name or id, returns an event clause to avoid that attendee when
      creating an event
VishayHai  # given a string, returns an event to match or create an event with that subject
IsSthanPar  # given a string, returns an event clause to match or create an event at that location
SePrarambh  # given a datetime clause, returns an event clause to match or create an event starting at
      that time
PeSamapt  # given a datetime clause, returns an event clause to match or create an event ending at that
      time
AvdhiHai  # given a time unit value, returns an event clause to match or create an event with that
      duration
SthitiHai  # given a ShowAsStatus value, returns an event clause to match or create an event with that
      status

# the following operators return datetime clauses and accept no arguments
DopaharBaad
Naashta
DerNashta
RaatKaBhojan
Jaldi
KaryaDivasSamapt
Shaam
MaahKaPuraMaah
VarshKaPurnaVarsh
PichleHafteNaya
Der
DerDopahar
DerSubah
DopaharKaBhojan
Subah
AglaMaah
AglaSaptahant
AgleHafteKiSuchi
AglaVarsh
Raat
Dopahar
Abhi
Patjhad
Vasant
Grishm
Shishir
IsHafte
IsSaptahant
Aaj
Kal
BitaKal

# general date time clauses
DinankSamayVarg  # given either a datetime clause representing a date and/or a time operator
      representing a time, returns a datetime clause
DinankVarg  # given a date or dayofweek, returns a date
SaptahKaVarshikDin  # given a day of week string, returns a time clause
AglaKaryaDiwas  # given a day of week string, returns a time clause for the next occurrence of that day
      of week
MahinaDin  # given a month and day as arguments, returns a date clause
MahinaDinVarsh  # given a month, day, and year as arguments, returns a date clause

# given a value, the following operators return datetime clauses according to the given value
MaahMein
ChaarAnkVarsh
GhantaMinatPoorvahn
GhantaMinatAparanh
SankhyaPoorvahn
SankhyaAparanh

# given a datetime clause, the following operators modify the clause and return a datetime clause
      according to the modification
DinankKeBaadSamay
DinankParSamaySePhele
SamayDinankKePaas

# given either a number or the operators EkDo/Kuch, all the following operators return time unit values
      according to the given unit
DinoMein
GhantoMein
MinatoMein

# these operators can be used to create time unit values instead of using integer values
EkDo
Kuch

SthitiDikhayein  # enumeration of possible event statuses (Busy, OutOfOffice)

Aur  # combines multiple event clauses together
```

Figure 9: Exhaustive list of operators for HI code generation (SMCalFlow).

```
TrouverÉquipeDe  # given a person name or id, returns a pseudo-person representing the team of that
      person
TrouverRapports  # given a person name or id, returns a pseudo-person representing the reports of that
      person
TrouverGestionnaire  # given a person name or id, returns the manager of that person

avec_participant  # given a person name or id, returns a clause to match or create an event with that
      person as an attendee
éviter_participant  # given a person name or id, returns an event clause to avoid that attendee when
      creating an event
a_sujet  # given a string, returns an event to match or create an event with that subject
Ã _emplacement  # given a string, returns an event clause to match or create an event at that location
commence_Ã   # given a datetime clause, returns an event clause to match or create an event starting at
      that time
se_termine_Ã   # given a datetime clause, returns an event clause to match or create an event ending at
      that time
a_durée  # given a time unit value, returns an event clause to match or create an event with that
      duration
a_statut  # given a ShowAsStatus value, returns an event clause to match or create an event with that
      status

# the following operators return datetime clauses and accept no arguments
AprÃ¨sMidi
PetitDéjeuner
Brunch
DÃ®ner
TÃ´t
FinDeJournéeDeTravail
Soirée
MoisEntierDuMois
AnnéeComplÃ¨teDeL'Année
NouvelleDerniÃ¨reSemaine
Tard
FinD'AprÃ¨sMidi
FinDeMatinée
Déjeuner
Matin
MoisProchain
WeekEndProchain
ListeProchaineSemaine
AnnéeProchaine
Nuit
Midi
Maintenant
Automne
Printemps
Été
Hiver
CetteSemaine
CeWeekEnd
Aujourd'hui
Demain
Hier

# general date time clauses
ClasseDateHeure  # given either a datetime clause representing a date and/or a time operator
      representing a time, returns a datetime clause
ClasseDate  # given a date or dayofweek, returns a date
ClasseJourDeSemaine  # given a day of week string, returns a time clause
ProchainJourOuvré  # given a day of week string, returns a time clause for the next occurrence of that
      day of week
MoisJour  # given a month and day as arguments, returns a date clause
MoisJourAnnée  # given a month, day, and year as arguments, returns a date clause

# given a value, the following operators return datetime clauses according to the given value
versMois
enAnnéeÀQuatreChiffres
HeureMinuteAM
HeureMinutePM
NombreAM
NombrePM

# given a datetime clause, the following operators modify the clause and return a datetime clause
      according to the modification
ÀDateAprÃ¨sHeure
ÀDateAvantHeure
AutourDateHeure

# given either a number or the operators EkDo/Kuch, all the following operators return time unit values
      according to the given unit
enJours
enHeures
enMinutes

# these operators can be used to create time unit values instead of using integer values
UnCouple
Quelques

AfficherCommeStatut  # enumeration of possible event statuses (Busy, OutOfOffice)

ET  # combines multiple event clauses together
```

Figure 10: Exhaustive list of operators for FR code generation (SMCalFlow).

```
НайтиКомандуДля  # given a person name or id, returns a pseudo-person representing the team of that person
НайтиОтчеты  # given a person name or id, returns a pseudo-person representing the reports of that person
НайтиМенеджера  # given a person name or id, returns the manager of that person
с_участником  # given a person name or id, returns a clause to match or create an event with that person as an attendee
избегать_участника  # given a person name or id, returns an event clause to avoid that attendee when creating an event
имеет_тему  # given a string, returns an event to match or create an event with that subject
в_местоположении  # given a string, returns an event clause to match or create an event at that location
начинается_в  # given a datetime clause, returns an event clause to match or create an event starting at that time
заканчивается_в  # given a datetime clause, returns an event clause to match or create an event ending at that time
имеет_продолжительность  # given a time unit value, returns an event clause to match or create an event with that duration
имеет_статус  # given a ShowAsStatus value, returns an event clause to match or create an event with that status


# the following operators return datetime clauses and accept no arguments
ПослеПолудня
Завтрак
Бранч
Ужин
Рано
КонецРабочегоДня
Вечер
ПолныйМесяцМесяца
ПолныйГодГода
НоваяПрошлаяНеделя
Поздно
ПозднийПолдень
ПозднееУтро
Обед
Утро
СледующийМесяц
СледующиеВыходные
СписокСледующейНедели
СледующийГод
Ночь
Полдень
Сейчас
Осень
Весна
Лето
Зима
ЭтаНеделя
ЭтиВыходные
Сегодня
Завтра
Вчера

# general date time clauses
КлассДатаВремя  # given either a datetime clause representing a date and/or a time operator representing a time, returns a datetime clause
КлассДата  # given a date or dayofweek, returns a date
КлассДеньНедели  # given a day of week string, returns a time clause
СледующийРабочийДень  # given a day of week string, returns a time clause for the next occurrence of that day of week
МесяцДень  # given a month and day as arguments, returns a date clause
МесяцДеньГод  # given a month, day, and year as arguments, returns a date clause

# given a value, the following operators return datetime clauses according to the given value
вМесяц
вЧетырехзначныйГод
ЧасМинутаАМ
ЧасМинутаПМ
ЧислоАМ
ЧислоПМ

# given a datetime clause, the following operators modify the clause and return a datetime clause according to the modification
НаДатуПослеВремени
НаДатуДоВремени
ОколоДатыВремени

# given either a number or the operators Acouple/Afew, all the following operators return time unit values according to the given unit
вДни
вЧасы
вМинуты

# these operators can be used to create time unit values instead of using integer values
Пара
Несколько

ПоказатьКакСтатус  # enumeration of possible event statuses (Busy, OutOfOffice)

СоздатьСобытие  # given multiple event clauses (such as с_участником, имеет_тему, combined together with `И`), creates an event complying with those clauses
НайтиСобытия  # given multiple event clauses (such as с_участником, имеет_тему, combined together with `И`), returns a list of events complying with those clauses
ТекущийПользователь  # returns the current user (person)

И  # combines multiple event clauses together
```

Figure 11: Exhaustive list of operators for RU code generation (SMCalFlow).

USER

```
original question: Add engagement party to monday 9 pm with Abby kim and desi

code: CreateEvent( AND( with_attendee( Abby ) , with_attendee( Kim ) ,
    with_attendee( Desi ) , has_subject( engagement party ) ,
    starts_at( NextDOW( MONDAY ) ) , starts_at( NumberPM( 9 ) ) ) )

decomposition:
```

ASSISTANT

```
{
  "Add engagement party to monday 9 pm with Abby kim and desi": {
    "code": "CreateEvent( AND( with_attendee( Abby ), with_attendee( Kim ),
      with_attendee( Desi ), has_subject( engagement party ),
      starts_at( NextDOW( MONDAY ) ), starts_at( NumberPM( 9 ) ) ) )",
    "decomposition": {
      "Add engagement party with Abby kim and desi": {
        "code": "CreateEvent( AND( with_attendee( Abby ),
          with_attendee( Kim ), with_attendee( Desi ),
          has_subject( engagement party ) ) )",
        "decomposition": {
          "Add event engagement party": {
            "code": "CreateEvent( has_subject( engagement party ) )",
            "decomposition": {}
          },
          "Add event with Abby kim and desi": {
            "code": "CreateEvent( AND( with_attendee( Abby ),
              with_attendee( Kim ), with_attendee( Desi ) ) )",
            "decomposition": {}
          }
        }
      },
      "Add event to monday 9 pm": {
        "code": "CreateEvent( AND( starts_at( NextDOW( MONDAY ) ),
          starts_at( NumberPM( 9 ) ) ) )",
        "decomposition": {}
      }
    }
  }
}
```

Figure 12: An in-context example for query decomposition.