Dual Model Error-of-Thought for Semantic Parsing

Anonymous ACL submission

Abstract

When evaluating Large Language Models (LLMs) with few-shot learning and Small Language Models (SLMs) on compositional generalization test sets of semantic parsing problems, we observe that each have different strengths but neither is clearly an improvement over the other. In this work we explore the possibility for combining the strengths of both approaches and mitigating possible negative interference. We show that framing the problem as error-correction from the SLM to the LLM can significantly improve performance above LLM and SLM baselines. We introduce a new method for error-correction of compositional generalization errors made by the SLM called Error-of-Thought (EoT). We show that EoT is simple to implement and nearly monotonically improves performance over both models.

		LLM + RAG					
		SMCalFlow_{32}		$\text{GeoQuery}_{32}^{\text{TMPL}}$		$\text{COGS}_{36}^{\text{vf}}$	
		✓	✗	✓	✗	✓	✗
SLM	✓	47.06	18.10	65.85	17.20	71.14	14.00
	✗	12.22	22.62	7.62	9.34	11.62	3.24

SMCalFlow	
Input	have my entire team attend a coding camp this weekend
SLM	`CreateEvent(time(ThisWeek()), subject("coding camp"),with("team"))`
LLM	`CreateEvent(subject("coding camp"),with(FindTeam(Me())), duration(days(7)))`
✓	`CreateEvent(time(ThisWeek()), subject("coding camp"),with(FindTeam(Me())))`

Table 1: **Above:** Confusion matrix (%) between fine-tuned T5 (SLM, vertical) and GPT-5 with SetBSR retrieval (LLM, horizontal) on the compositional generalization test splits of three common semantic parsing datasets. A significant number of cases are correct in one or the other approach, but not both. Results for other datasets shown in Appendix J. **Below:** an example of how SLM and LLM responses can differ. The SLM often gets the **compositional** case wrong due to overfitting, the LLM can get the compositional case correct, but misses **other details** which couldn't be easily retrieved.

1 Introduction

A common feature of both human and code-like languages is so-called "compositionality", where complex expressions are made by combining together simple expressions, and where simple expressions may be substituted within their class to make a new complex expression where only that corresponding part of the meaning has changed. This allows for the generation of an infinite amount of meanings by substituting parts (Chomsky, 1957).

In a human language such as English, we can start from the expression "I'd like to create a meeting on Wednesday with Joan" and form the expression "I'd like to create a meeting after lunch with my supervisor". The intent remains constant, but the time Wednesday is swapped for "after lunch" ands the meeting attendee is the user's manager. Likewise in code languages that express these concepts, we might encode the expressions as a series of nested function calls, where the first would be `CreateEvent(with=Person("Joan"), time=Day("Weds"))` and the second might be `CreateEvent(with=FindBoss(Person(Me())), time=After(Lunch()))`, with similar constant form but different atoms.

In the *semantic parsing* problem, the goal is to find some way to translate human instructions into a corresponding code representation for deterministic processing by another system. Modern machine translation methods using neural language models can do this quite well, but they have well-known shortcomings when it comes to handling compositional generalization (Hupkes et al., 2020). The performance of a model is usually determined by the diversity of training data, but if a

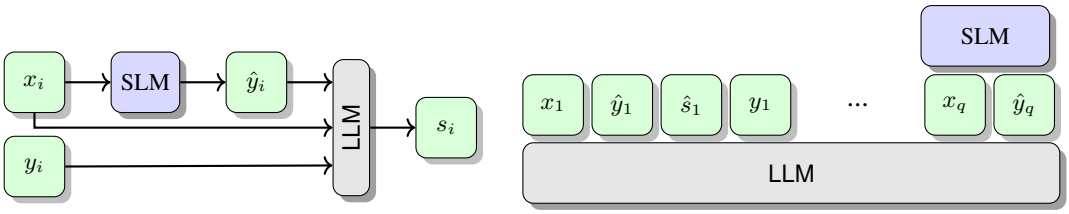

Figure 1: Error-of-Thought (**EoT**) Dataflow. The system works by generating error identification and fix instructions based on partially correct generations from a different model fine-tuned on part of the problem domain. **Left**: Construction of the error reasoning s_i for each example, by using the SLM to generate an answer $\hat{y}_i$ for a possibly OOD x_i from few-shot data $\mathcal{D}_{\text{comp}}$, then comparing with the gold y_i using an LLM. **Right**: Usage of EoT at inference time, by first providing examples of collected incorrect answers and their explanations and true answer, then by generating a possibly incorrect answer $\hat{y}_q$ for a given OOD x_q using SLM and using the LLM to generate the analysis s_q and final answer $\hat{y}_q$. The system differs from ERP since SLM is used to generate s_i instead of an LLM, and from CoT since s_i contains *error correction* instructions from $\hat{y}_i$ to y_i and not step-by-step instructions to convert from x_i to y_i.

model's training data only included a small number of samples for a particular subset of compositional behavior $\mathcal{D}_{\text{ood}}$, then the subset's performance will be poor, even if the individual *parts* are well represented in the training data. The failure mode is illustrated by the SLM response in Table 1, where the SLM writes `with("team")` instead of `with(FindTeam(Me()))`. This drawback is particularly relevant in semantic parsing because much of the point is to support compositional requests or extension with few-shot data.

The failure stems from the sequence maximum-likelihood training objective which makes neural generative models so good at solving the in-distribution problem in the first place:

$$\arg\max_{\theta} \log P(x_{t+1}|x_t, ..., x_0) \quad (1)$$

This objective maximizes confidence on the right answer, but minimizes confidence on incorrect and out-of-distribution (OOD) compositions later on (Li, 2025). Past works try to handle this problem with data augmentation (Yao and Koller, 2024; Qiu et al., 2022; Yang et al., 2022b; Akyürek et al., 2021; Andreas, 2020), but that is difficult to do well without knowledge of the dataset rules and what to generate. Without information on what the future compositions may be, there is no straightforward way to find a θ that satisfies the objective in both the training data and and the OOD test data.

1.1 LLMs and In-Context Learning

In contrast to standard fine-tuning of language models, large language models (LLMs) are trained on web-scale text but with a similar sequence maximum-likelihood task. The model parameter count and training data sizes are orders of magnitude larger than where we might fine-tune on an existing training data set. LLMs can also be used for semantic parsing, but usually they would be used with some additional examples placed into the input-context $(x_t, ..., x_0)$. This is often referred to as "In-Context Learning" (ICL). Typically ICL is used with in combination with a retrieval system applied over a larger dataset, in a system called Retrieval Augmented Generation (RAG). The CEIL (Ye et al., 2023), SetBSR (Gupta et al., 2023) and GandR (Zemlyanskiy et al., 2022) retrieval methods have shown some promise in improving downstream performance by careful selection of the examples. Very long context models which admit millions of tokens in the context even permit the entire retrieval set or at least a large sample of it to be placed in the context (Agarwal et al., 2024).

LLMs can also be used in a "thinking" or even "agentic" manner to perform semantic parsing. In this case the LLM has already been fine-tuned to replicate a step-by-step thinking process and an instruction following-process (Wei et al., 2022; Zhou et al., 2023). By providing a sufficient description of the problem or rules (Yang et al., 2022a; Wang et al., 2023), the LLM can simulate a process of putting together the semantic parse piece-by-piece. Alternatively, we can use an error correction process, either by learning to self-correct (Li et al., 2025a), or learning a separate reasoning step corrector (Li et al., 2025b; Yang et al., 2024).

The biggest challenge with using LLMs is that they require a sufficiently detailed explanation of the problem or good examples to refer to for each problem instance. Selecting examples is non-trivial (Mitchell et al., 2021) and even optimal exam-

ples might not be easily searchable from the inputs (Spilsbury et al., 2024). Inducing the explanations from data alone can be challenging if the output language is nuanced (Wang et al., 2023). LLMs also have well-known problems with so-called "hallucination", where the decoding process can sometimes confidently produce incorrect plausible-looking text.

1.2 Error Correction with LLMs and SLMs

The disjoint set of advantages and drawbacks to fine-tuning SLMs and using in-context learning LLMs is also reflected in their diverging performance on reference semantic parsing datasets. To explore this problem we run a simple test by evaluating a strong retrieval augmented generation method, SetBSR on `gpt-5-mini` (LLM) and fine-tuned T5 (SLM) across three well-known semantic parsing datasets with compositional generalization test sets. We check the intersection, union and disagreement amongst correct answers and show the results in Table 1. Across all three datasets, between a quarter and a third of test instances fall in the *disagreement* cells , cases where one method is correct and the other is not. On SMCalFlow, 18.1% of instances are solved by the SLM alone and 12.2% by GPT-5 alone, giving a total disagreement rate of 30.3%. GeoQuery and COGS show similar patterns at 24.8% and 25.6% respectively. These disagreement rates show potential for improvement: if we could always select the correct answer when at least one method succeeds, the oracle upper bounds would reach 77.4% on SMCalFlow, 90.7% on GeoQuery, and 96.8% on COGS – well above either individual method.

Our intuition and hypothesis is that LLMs with few-shot prompting can solve the compositional problems easily but can get the details of the output language wrong. SLMs are specialized on the details, but this high level of specialization loses the compositional generalization ability.

Can we get the best of both worlds? The most obvious approach is to fine-tune with LoRA, but that requires both access to the model weights and care to not accidentally destroy the LLM's other abilities. We could also try to run both models at the same time, either by averaging their logit distributions (Zhao et al., 2024), or by deciding which one to trust (Shen et al., 2024), but this requires having white-box access to the model's decoding process – typically not available for high quality LLMs accessible via API. Another active area is *correction*, where the LLM generates candidates and we either try to rank which answer is best (Xiang et al., 2023) or correct the response, either by using domain specific error corrector (Gong et al., 2025; Zhu et al., 2026), or with generic reasoning based error correctors like Error Reflection Prompting (ERP, Li et al. (2025a)).

1.3 Contributions

In our work, we take a similar approach to ERP – but instead of self-reflecting on errors made by the LLM itself, we correct errors made by the fine-tuned SLM with an LLM. We contribute:

1. Dual Model Error-of-Thought (EoT) - An approach for combining the precision of SLMs with the generalization capability of LLMs.
2. Comprehensive evaluation of our method and against strong baselines on numerous well-known semantic parsing datasets.
3. Analysis of the strengths and weaknesses of EoT and where it is appropriate to use.
4. Qualitative comparison of EoT prompting compared to other prompting methods.

2 Method

We assume that our training dataset $\mathcal{D}$ can be partitioned into $\mathcal{D}_{\text{id}}$ and $\mathcal{D}_{\text{comp}}$, containing a small number (tens) of examples demonstrating the compositional cases.

Similar to Chain of Thought prompting, Error-of-Thought prompting places examples of error detection and correction in the LLM context. The model making the initial guess in this case is the SLM trained on $\mathcal{D}_{\text{id}}$. After making generations $\hat{i}_i$ for each x_i on $\mathcal{D}_{\text{comp}}$, we use the LLM to identify and summarize errors (s_i) when comparing to the ground truth y_i. The error-of-thought prompt S is then comprised of $x_0, \hat{y}_0, s_0, y_0, ...x_k, \hat{y}_k, s_k, y_k$, plus some general instructions (Figure 1).

At inference time, we first use the SLM to guess the answer $\hat{y}_q = \text{SLM}(x_q)$. Then we provide both $\mathcal{S}$ and $\hat{y}_q$ to the LLM and ask for both s_i and a correction: $\tilde{y}_q = \text{LLM}(\mathcal{S}, x_q, \hat{y}_q)$.

We answer three questions with this method:

1. Does the error correction procedure meaningfully improve upon the answers given by the SLM when it comes to semantic parsing?
2. Can error correction be a viable and competitive alternative to few-shot learning or grammar learning methods?
3. What kind of $\mathcal{S}$ is most suitable?

	SMC_8	SMC_{16}	SMC_{32}	GQ_{32}^{TMPL}	GQ_{32}^{TMCD}	$COGS_{36}^{vf}$	CFQ_{32}	$ATIS_{32}$	$CUDON_{32}$	mT_{32}
EoT (ours)	**57.01**	**67.42**	**76.3**	72.48	**81.61**	**93.46**	**74.40**	63.76	**72.24**	**43.67**
ERP	27.70	54.56	59.1	52.57	71.81	80.02	15.80	42.85	58.4	34.5
CoT	35.7	47.5	58.3	52.08	70.09	78.19	14.55	44.92	64.00	35.49
CASPER	26.71	34.45	56.47	75.92	66.18	78.35	35.90	62.52	56.65	24.64
SetBSR	41.3	50.3	59.2	73.46	77.74	81.01	21.09	55.72	66.25	40.13
GandR	52.03	61.2	66.2	74.69	80.63	85.43	23.01	58.38	65.50	40.08
Rerank	36.3	44.94	60.33	64.12	64.46	75.19	12.00	43.89	61.2	38.38
GPT-5	34.3	43.9	56.2	61.1	63.97	73.90	11.50	40.99	64.00	33.6
SLM-FT	29.7	44.7	59.8	**83.04**	63.72	82.67	67.8	63.56	52.3	25.6

Table 2: Comparison of string exact-match accurracy on various semantic parsing datasets. Due to experiment costs, only one experiment was run for each, with a fixed random seed over all experiments. The subscript refers to the number of few-shot examples in $\mathcal{D}_{\text{comp}}$ available to the method. Dataset and method abbreviations explained in Section 3.1 and C.7 Best result for each dataset **bolded** if it was at least 1%pt better than the next best baseline.

3 Experiments

3.1 Datasets

We provide brief descriptions of each dataset and modifications made. Details in Appendix G.

SMCalFlow-CS (SMC) (Andreas et al., 2020; Yin et al., 2021) is dataset consisting of natural language utterances and corresponding lispress programs for creating and managing calendar events. In the Compositional Splits (CS) dataset, the training set contains examples of creating events and finding people on an organizational chart, but only k examples of combinations of the two. The compositional problem is quite straightforward - references to a person's manager, reports or team need to be specified with a `Find*` call instead of a string. Uniquely, SMCalFlow-CS is one of the few datasets for studying compositional generalization that does not contain synthetically generated inputs. We use the version from Yin et al. (2021).

GeoQuery (GQ) (Zelle and Mooney, 1996) are synthetic English-like queries mapped to a logical predicate language for a U.S. geography database. The predicates can be deeply nested, so for example the English sentence "which states contain at least one major river" corresponds to `state(loc_1(major(river(all)))))`. The compositional splits introduced in (Finegan-Dollak et al., 2018; Shaw et al., 2021) which follow the "maximum compound divergence" (TMCD) and template methdology. In the template split, entities are first anonymized and the data is stratified such that no logical form appears in both the training and test sets. In the TMCD split, the divergence of compound distributions is maximized. We "deanonymize" the split dataset by substituing the entity names back after the split has been made.

COGS (Kim and Linzen, 2020) is a diagnostic dataset for various cases of natural language composition, for example active to passive. We use the "variable free" (vf) version of the dataset from Qiu et al. (2022), which is simplified to remove numbered positions referring to words in the input sequence. Because the test set for COGS is very large, we evaluate only on 1000 examples. These are balanced such that we have 50 examples from each labeled case in the original COGS paper.

CFQ (Keysers et al., 2020) which stands for Compositional Freebase Questions a synthetically generated language-to-SPARQL dataset in the cinema domain. The test splits are also based on the MCD principle. We evaluate on the first of three MCD splits one due to evaluation cost. The test for CFQ, as well as ATIS, CUDON and mTOP are all limited to 1000 randomly chosen examples.

ATIS (Price, 1990) consists of language utterances mapped to a DSL for querying flight information. Similar to GeoQuery and COGS, the language utterances are synthetically generated, but the order of words in the query does not necessarily correspond to the order of items in the output.

CUDON (Zheng et al., 2023) is made up of Mandarin Chinese requests paired with a DSL for searching stock-brokerage information. The test sets are made using the MCD principle. There are six MCD train/test splits. We evaluate on the TMCD3 split, which has a target compound divergence score of $\tilde{0}.21$.

mTOP (mT) (Li et al., 2021) is a multi-lingual semantic parsing dataset consisting of language utterances mapped to nested "intents". We consider the sparse training data case, with 1000 examples from the training set for T5 and for retrieval.

3.2 Baselines

We focus our comparison to retrieval, generation and prompting baselines. Particularly in the domain of semantic parsing there are many prior works which explore augmentation or constrained generation with grammar and alignment priors (Herzig and Berant, 2021; Qiu et al., 2022). These works can achieve very high scores, but in the context of our evaluation, they are not directly comparable as we try to minimize the assumptions made about each dataset. Further details in Appendix H.

Finetuning (SLM-FT) In this case, we simply fine-tune CodeT5 on the provided training data inputs and outputs. Details are in Section 3.3

LLM with few-shots We provide k shots (indicated in the dataset subscript) from the compositional set to the LLM as context for each test input in the case of GPT-5. In **Rerank**, we take best of 5 different generations from the LLM by SLM overall log-likelihood. In **CoT** we use k-shot chain-of-thought, where CoT examples are automatically generated by a prior LLM preprocessing step (Zhang et al., 2023). In all cases where LLMs are used the prompt requests reasoning first.

Retrieval Augmented Generation We create a vector index by encoding the training set pairs using sentence-transformer's `all-MiniLM-L6-v2` model. In **SetBSR** we retrieve from the index using the input, then use the SetBSR algorithm to rank and select up to 16 examples by set-coverage of the input tokens. In **GandR**, we first generate a candidate output using the **FT** model, then search the index using both the input and output. These examples are added to the k-shots for GPT-5.

Generation and Reflection In **CASPER** we use the method in Pasupat et al. (2021); Spilsbury et al. (2024) to generate new examples using parts of the query and the provided few-shot examples. **ERP** is the closest method to EoT. In this method we generate error correction instructions by evaluating the LLM on each of the few-shot inputs along with its retrieved neighbours from SetBSR.

3.3 Setup

The subscript in the dataset name indicates that number of few-shot examples demonstrating the compositional generalization domain. A subscript in the method name indicates that n-fewshot examples are either retrieved or generated based on $\mathcal{D}_{\text{id}}$ in addition to the generalization fewshots.

Where it is necessary to train an SLM, we train for 100 epochs on the training and few-shot data with per-batch balanced sampling. We use the T5-small model architecture, starting from the `salesforce/codet5p-220m` checkpoint. The learning rate is 0.001 and the batch size is 128. The entropy bonus weight τ is set at 0.1.

The LLM used in all Table 2 evaluations is the `gpt-5-mini` by OpenAI. Prompts used depend on method but all prompts include general instructions to follow syntax and argument ordering provided in examples carefully and to provide reasoning prior to answering. See Appendix C for detailed information on the prompts used. For EoT, the prompt includes a general description of errors inducted from evaluations of the fine-tuned SLM on the few-shot data, then for each few-shot example, `gpt-5-mini`'s guess of which errors were detected from the SLM response prior to showing the gold output. We use the same fine-tuned SLM as in **SLM-FT** to make the $\hat{y}_i$ and $\hat{y}_q$ predictions for the LLM input, but fine-tuning excludes the k few-shot examples. We also include estimated back translations from $\hat{y}_i \rightarrow \hat{x}_i$ using an SLM trained on reversed pairs as an additional hint to detect possible errors. Other compared baselines don't have the same initial guess $\hat{y}$ provided.

Like prior works, our metric is exact match accurracy. In general EoT maintains competitive performance against all other retrieval, generation and prompting based baselines. The strongest gains are seen in SMCalFlow, COGS, CFQ and CUDON. EoT does not outperform a fine-tuned SLM on the Template split of GeoQuery. We suspect the reason for this is that GeoQuery benefits much more from retrieval methods, as explored in Section 3.4. Our results show that our method is a viable and robust mechanism to incorporate and improve upon SLM responses using an LLM and can perform in basically all cases either as well or better than both LLM-only and SLM- only baselines.

	SMC_{32}	GQ_{32}^{TMPL}	$COGS_{36}^{vf}$
Oracle + EoT	87.9	86.97	94.57
ERP + SLM $\hat{y}$	73.2	68.05	81.8
CoT + SLM $\hat{y}$	67.06	73.48	96.77
EoT w/o SLM Guess	61.23	56.51	81.28
EoT + RAG	75	83.29	96.9
Error Ex. + SLM $\hat{y}$	67.2	74.93	91.71
Directions + SLM $\hat{y}$	65.23	69.87	95.52
Only Directions	61.2	50	84.9
Grammar Prompt	65.7	80.3	74.91

Table 3: Different types of $\mathcal{S}$ and performance with GPT-5 mini. Oracle + EoT correponds to Section 3.10, the next three rows correspond to Section 3.6. EoT + RAG includes retrievals from SetBSR. Directions and Grammar prompt don't include errors.

	SLM → EoT		SLM → RAG	
	✗ → ✓	✓ → ✗	✗ → ✓	✓ → ✗
SMC_{32}	38.9	1.12	21.11	4.07
GQ_{32}^{TMPL}	46.03	22.67	11.30	12.28
$COGS_{36}^{vf}$	79.58	3.17	15.84	4.65

Table 4: Conditional error analysis of correcting SLM responses with EoT versus replacing SLM with LLM-RAG (SetBSR) responses. Numbers are as percentages of cases in the original set. A good method will correct all errors (approach 100% on ✗ → ✓) without introducing new errors on already correct items (0% ✓ → ✗).

3.4 Analysis

We dissect further where EoT's strengths and limitations are and how it can best be used.

3.5 Error correction beats rule prompts

We can imagine other types of problem-assisting problems that we can induce from the few-shot data. For example (Wang et al., 2023) induce a BCNF grammar from few-shot data, then construct prompts to guess the relevant parts of the grammar and alignment for each example (Grammar Prompt). This can also be done on a more abstract level, by using the few-shot data to induce "directions" on how to convert inputs to outputs (Directions). Error Ex. corresponds to including only inputs, $\hat{y}_i$ and a correct indicator without analysis.

In all three cases, we observe that the performance on SMCalFlow is worse than EoT and ERP, even though the methods do match or improve upon the retrieval baselines. We suspect that a part of the challenge is that the semantic parsing problem can be quite complex and have a lot of edge cases, or cases that aren't easily capturable or describable in rules. There are many more failure cases when trying to come up with this as opposed to detecting common error patterns. We note that Wang et al. (2023) were only able to get good performance once they had hand-specified a comprehensive "oracle" grammar and alignment rules in the prompt. While error prompting does not quite meet the same level of performance as prompting with the "oracle" grammar, it comes very close and does not require the same level of human effort.

We also see that the analysis and summarization of the errors is important as demonstrated by the gap in performance between EoT and Error Ex., even when both have the SLM guess available.

3.5.1 Prompting for the right errors matters

ERP is the closest related work to EoT. The two main differences are that in ERP the error prompt is constructed using errors made by an LLM over $\mathcal{D}_{\text{comp}}$ (compared to errors made by the SLM in EoT) and there is no "SLM guess" ($\hat{y}_q$) part of the prompt. To isolate the impact of using the SLM to find errors, we add the SLM guess $\hat{y}_q$ to the ERP prompt. Intuitively, we would expect that errors corresponding to the model that made the guess will be more informative than errors coming from a different model. Comparing EoT in Table 2 and ERP + SLM $\hat{y}_q$ in Table 3, we find a difference of a few percentage points. This roughly aligns with our intuition, but it does also suggest that the errors made by the few-shot model have some degree of overlap with the errors made by the SLM. We explore in more depth in Appendix F.

3.6 Initial guess ($\hat{y}_q$) from the SLM helps a lot

Since ERP does not make an initial guess for a test example, we compare ERP with ERP + SLM $\hat{y}$. Intuitively, we expect that the initial guess helps, because if reduces the work for the LLM to error analysis and correction as opposed to also generating the output sequence from scratch. For ERP and EoT we find that having the initial guess by the SLM helps significantly, providing a 13 point boost on SMCalFlow in the case of ERP and a 10 point boost in the case of GeoQuery. We find as well that providing the example helps even when the prompt is CoT as opposed to EoT, on SMCalFlow this boosts performance by about 4 points.

3.7 Monotonic improvement on SLM guesses

Even though EoT generally improves on both the best fine-tuning and RAG baselines in overall per-

formance, it could be that it introduces errors in areas where those methods were previously correct and generates correct answers in other areas to “compensate". If this were the case, then overall performance improvement would be less meaningful. To analyze if this is the case we perform conditional error analysis in Table 4 (and further qualitative and quantitative checks in Appendices E K). In general we find that for SMCalFlow and COGS, the EoT method corrects a large share of T5 errors and introduces few errors of its own, except in the case of GeoQuery where 22% of errors are newly introduced. The improvement on SetBSR is less pronounced - EoT corrects 10 to 20% of RAG errors and 4 to 10% of errors are newly introduced ones over RAG.

3.8 RAG might not be needed

Since the performance gain coming from EoT outpaces that of RAG methods like SetBSR and GandR, we ask whether EoT can substitute those methods. We evaluate EoT both without RAG (in Table 2) and with RAG in Table 3. For SMCalFlow, performance is actually marginally improved *without* RAG, but with there is a 11 point drop on GeoQuery (below that of finetuning T5) and a 2 point drop in the case of COGS. The fewshot example selection strategy likely has something to do with this. In SMCalFlow, the few-shot data shows the main idea behind how compositional generalization should happen. But for GeoQuery and COGS, we sample 32 examples randomly from the generalization test data, which may not sufficiently explain how the data distribution differs. This can be corrected by simply adding RAG (in the form of SetBSR) back into the prompts. After doing so, EoT is able to at least match or better yet outperform both fine-tuning T5 and the RAG methods.

3.9 Performance scales with the number of few-shots

Like other few-shot methods, the performance of EoT scales corresponding to the number of examples. This can be seen in the evaluation of the SMCalFlow dataset with 8, 16 and 32 few-shots. We saw 10 point gains for each doubling of the number of shots, but this gain likely plateaus, similar to the observations in (Rajkumar et al., 2022).

3.10 Overcorrection risk if the SLM is perfect

If the SLM was perfect there would be no need for EoT, as we could always use the SLM answers. The error prompt $\mathcal{S}$ would also be empty as there would be no errors on $\mathcal{D}_{\text{comp}}$. Nevertheless, it is worth understanding the worst-case behaviour when EoT induces false positives. To study this, we consider an “Oracle" SLM, which is always provides the correct initial answer to the LLM as part of the EoT process. The LLM receives the same error set $\mathcal{S}$ as it would have before. Degradation in performance from 100% indicates cases where EoT falsely detected errors and applied unnecessary fixes.

The results are indicated in Table 3 corresponding to the line “Oracle". For SMCalFlow, EoT has an error rate of about 12% and for COGS an error rate of 5.5%. It is notable that the COGS Oracle EoT error rate is actually marginally *higher* than the COGS Non-Oracle EoT error rate. This at least indicates that a mismatch between the model which produces the error prompt and the model providing the guess $\hat{y}_q$ can result in suboptimal performance.

	SMC_{32}	$\text{COGS}^{\text{vf}}_{36}$
EoT	81.72	75.4
SetBSR	63.59	86.9
SLM	83.68	100

Table 5: Evaluation on in-distribution test data. Using the T5 model trained on the same training distribution and (irrelevant) error prompts from the few-shot examples in the case of EoT. SetBSR includes retrievals from the training data and compositional few-shot examples.

3.11 OOD error correction harms ID performance slightly

We wish to test the robustness of our method also when evaluated on in-distribution data. Adding few-shot examples or error prompts for data from a different distribution may degrade performance when evaluated in-distribution. To test this case, we checked the two datasets which provide separate in-distribution test – SMCalFlow and COGS, but with models that have been “exposed" to the compositional generalization distribution through the few-shot data. In SetBSR, the few-shots are included as part of the prompt and for EoT they are included in the prompt error corrections. We find that both methods perform worse than the SLM on ID data, but on SMCalFlow EoT is able to preserve most of the performance, with only a 2%pt difference in absolute performance and a 3.3% “overcorrection" rate. On COGS the results are not as strong - EoT introduces an error rate of about 25%, though in this case the T5 model was 100% correct.

	Dataset	GLM	Gemma3	Qwen3.5
EoT	SMCalFlow_{32}	57.16	71.49	72.33
	GeoQuery	58.9	74.69	61.65
	COGS	65.85	83.33	68.76
CoT	SMCalFlow_{32}	36.50	50.22	46.73
	GeoQuery	38.57	61.67	51.11
	COGS	56.19	31.61	24.87
SetBSR	SMCalFlow_{32}	41.99	48.56	68.33
	GeoQuery	60.1	63.39	69.04
	COGS	32.86	43.15	54.79
FT+RAG	SMCalFlow_{32}	15.53	62.89	18.10
	GeoQuery	56.02	61.67	57.98
	COGS	52.95	75.52	18.53

Table 6: Exact-match scores of EoT compared to SetBSR and CoT prompting on different LLMs.

Condition	SMC_{32}	$\text{GQ}_{32}^{\text{TMPL}}$	$\text{COGS}_{36}^{\text{vf}}$
Missed Correction	7.99	0.91	0.1
Missed Composition	2.56	1.14	0.9
Overcorrect Add	5.58	18.95	1
Overcorrect Remove	2.56	3.88	0.1
Overcorrect Reorder	2.11	0	0.4
Formatting	2.86	0	0.7
No errors	76.319	75.11	96.8

Table 7: Error analysis (% of test data) for EoT. A missed item is one where the SLM produced and incorrect output and EoT failed to correct it. A overcorrection is where the SLM answer was already correct or partially correct, but EoT introduced a new error.

3.12 EoT improves performance across different LLMs, but parity with a fine-tuned SLM remains elusive.

We also expand our evaluation beyond GPT-5-mini by testing on other open-weight LLMs. In general we found that it was quite difficult to get good performance from the smaller open-weight LLMs, so the results differ quite significantly from the GPT-5-mini case. On Qwen and GLM, we had to disable the thinking mode, otherwise the model would get stuck in a thinking loop. Further details of the LLM setup are in Appendix I. For SMCalFlow, we used the simplified dataset version (Meron, 2022).[1] EoT improves upon SetBSR (even a LoRA fine-tuned SetBSR) in almost every case, except for GeoQuery on thinking models, although it does not reach the performance of a fine-tuned SLM, meaning that there is significant overcorrection. This effect is worse on the thinking models like Qwen and GLM.

3.13 EoT failure cases are typically overcorrections or non-compositional errors made by the SLM.

We inspect the test examples that EoT got wrong and summarize the categories in Table 7. The categorization was done by manually inspecting the outputs with the help of an LLM generated summary of what the in question was. First we consider cases where EoT failed to correct SLM errors (denoted as 'Missed'). A *Missed Correction* is one where the error was **not** a compositional generalization error, but was instead a parsing error by the SLM which would have occurred on similar in-distribution data. For example in the case of SMCalFlow, the input text might have requested a meeting "excluding Abby" but the SLM instead added Abby as an explicit attendee. A *Missed Composition* is a compositional generalization error, for instance leaving "meeting with my team" in the `:subject` area instead of adding an attendee by `FindTeamOf(CurrentUser())`.

Overcorrections are cases where EoT corrected any SLM errors, or perhaps there were no SLM errors, but then new errors were introduced by the correction process. *Add* denotes when the EoT process added new incorrect output erroneously and *Remove* denotes when correct elements were removed. *Reorder* and *Formatting* are cases when the output contains all the necessary elements and don't contain any incorrect ones, but is just in the wrong order or is missing syntax elements like parentheses. These are explained further in Appendix D. We find that most of the EoT errors can be explained by the reorder and formatting cases, which indicates that future work could focus on finding a way to mitigate this overcorrection.

4 Conclusion

In this work we presented Dual Model Error-of-Thought prompting and provided comprehensive evaluation and analysis on compositional generalization splits of semantic parsing datasets. Error-of-Thought when applied correctly can monotonically boost performance above SLM and LLM baselines. We showed that most of EoT's advantage comes from having the SLM initial guess, such that the LLM corrects only the systematic errors that overfitted SLMs might make in the compositional generalization case.

[1] For reference, T5 scores 62.89 on this dataset version

Limitations

The limitations of this work are as follows.

First, our experimental evaluation was limited by budgetary constraints. OpenAI's closed-weight models can achieve very good performance with correct prompt design, but they are also expensive to use at scale. This limited the extent of our evaluation, For example, we were not in a position to run the same experiment multiple times with different seeds to report confidence intervals etc. We also needed to limit the size of test sets to something reasonable, instead of evaluationg on tens of thousands of test examples.

Second, many semantic parsing datasets don't include a specific few-shot partition of the dataset for their compositional generalization cases – they only consider zero-shot compositional generalization. Our method assumes that some few-shot data is available. In order to make the experiments work with our method, we randomly sampled few-shot data from the test-set and then excluded that data from test evaluation. In our experience, this few-shot data isn't optimal for explaining the domain transfer between $\mathcal{D}_{\text{id}}$ and $\mathcal{D}_{\text{ood}}$ in the same way that datasets with pre-provided few-shot data like SMCalFlow have.

Third, because we have modified the datasets slightly (though not making any assumptions about its structure), all of our baselines are re-implementations of existing published baselines as opposed to using previously published results. We have made these re-implementations on a best-effort basis, though they may not work exactly the same as in the original works.

For these three reasons, **we don't claim that our results are state-of-the-art** on any particular dataset in any particular evaluation setting, but rather we use our experiments to demonstrate that EoT prompting can significantly improve upon the strongest baseline results that we were able to achieve using known methods.

The Error-of-Thought method clearly has drawbacks in certain situations as explored in sections 3.10 and 3.11. Like any few-shot method, it is also sensitive to the choice of few-shot data used to build the initial errors prompts. So for this reason, any real-world deployment needs to consider the few-shot data carefully to make sure that it really represents the real error case distribution.

Finally, our method relies on large language models for providing the final answer. It is quite well known that these large language models are both expensive to run and have high energy consumption requirements. We established in Section 3.12 that while EoT does help performance on smaller LLMs, it still doesn't beat the baseline of just using a small fine-tuned LLM on the in-distribution data in many cases. This makes the work less practical for deployment within for example embedded settings. A possible future work could be to look at distillation so that an error-corrected model can work in a constrained resource setting.

Ethical Considerations and Impact

Our work is foundational in nature and we do not anticipate any specific ethical risks to arise directly from our work. We advise readers that compositional generalization and out-of-distribution generalization more broadly is still an active research area and contains many nuances that are not well understood. In addition, our work relies on foundational and commercial LLMs, which by now have their own widely understood ethical considerations and risks around their use – particularly around the issues of energy consumption, training dataset and/or alignment bias and the "hallucination" problem which sometimes results in confident but inaccurate outputs. The results in our paper show that in many cases LLMs can improve on initial guesses made by an SLM trained on in-domain data, but this is not automatic and does not happen in every case. For these reasons, care and careful consideration of whether the problem in question is sufficiently similar to the research problem in this paper is advised before applying this work or other related works to industrial or safety-critical settings.

Computational Resource Usage

We ran experiments on our internal compute cluster. Each experiment takes around 5 hours to complete on a typical NVIDIA datacenter GPU. For cases where we used OpenAI, we used the standard OpenAI API. Total API costs were about USD$500 and were funded by the authors.

Supplementary Materials

We provide JSON files containing the overall test results and generations with each experiment as well as log files from our cluster which show the individual LLM calls and their inputs

and outputs. We also provide a copy of the code used to run the experiments. Since all of these files together exceed the file size limit on OpenReview, we have made it available at https://acl-arr-error-of-thought-paper.s3.us-east-1.amazonaws.com/supplementary-results.zip.

References

Rishabh Agarwal, Avi Singh, Lei Zhang, Bernd Bohnet, Luis Rosias, Stephanie C. Y. Chan, Biao Zhang, Ankesh Anand, Zaheer Abbas, Azade Nova, John D. Co-Reyes, Eric Chu, Feryal M. P. Behbahani, Aleksandra Faust, and Hugo Larochelle. 2024. Many-shot in-context learning. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Ekin Akyürek, Afra Feyza Akyürek, and Jacob Andreas. 2021. Learning to recombine and resample data for compositional generalization. In *International Conference on Learning Representations*.

Jacob Andreas. 2020. Good-enough compositional data augmentation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7556–7566. 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, and 26 others. 2020. Task-oriented dialogue as dataflow synthesis. *Trans. Assoc. Comput. Linguistics*, 8:556–571.

Noam Chomsky. 1957. *Syntactic Structures*. Mouton and Co., The Hague.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir R. Radev. 2018. Improving text-to-sql evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 351–360. Association for Computational Linguistics.

Yue Gong, Chuan Lei, Xiao Qin, Kapil Vaidya, Balakrishnan Murali Narayanaswamy, and Tim Kraska. 2025. SQLens: An end-to-end framework for error detection and correction in text-to-SQL.

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, Singapore, December 6-10, 2023*, pages 13924–13950. Association for Computational Linguistics.

Jonathan Herzig and Jonathan Berant. 2021. Span-based semantic parsing for compositional generalization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 908–921. Association for Computational Linguistics.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. 2020. Compositionality decomposed: How do neural networks generalise? *J. Artif. Intell. Res.*, 67:757–795.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. 2020. Measuring compositional generalization: A comprehensive method on realistic data. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Najoung Kim and Tal Linzen. 2020. COGS: A compositional generalization challenge based on semantic interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 9087–9105. 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, EACL 2021, Online, April 19 - 23, 2021*, pages 2950–2962. Association for Computational Linguistics.

Jason Li, Lauren Yraola, Kevin Zhu, and Sean O'brien. 2025a. Error reflection prompting: Can large language models successfully understand errors? In *The Sixth Workshop on Insights from Negative Results in NLP*, pages 157–170, Albuquerque, New Mexico. Association for Computational Linguistics.

Yiqi Li, Yusheng Liao, Zhe Chen, Yanfeng Wang, and Yu Wang. 2025b. DICE: structured reasoning in llms through slm-guided chain-of-thought correction. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pages 6951–6966. Association for Computational Linguistics.

Yuanpeng Li. 2025. A theoretical analysis of compositional generalization in neural networks: A necessary and sufficient condition. *ArXiv*, abs/2505.02627.

Joram Meron. 2022. Simplifying semantic annotations of SMCalFlow. In *Proceedings of the 18th Joint ACL - ISO Workshop on Interoperable Semantic Annotation within LREC2022*, pages 81–85, Marseille, France. European Language Resources Association.

Eric Mitchell, Chelsea Finn, and Christopher D. Manning. 2021. Challenges of acquiring compositional inductive biases via meta-learning. In *AAAI Workshop on Meta-Learning and MetaDL Challenge, MetaDL@AAAI 2021, virtual, February 9, 2021*, volume 140 of *Proceedings of Machine Learning Research*, pages 138–148. PMLR.

Panupong Pasupat, Yuan Zhang, and Kelvin Guu. 2021. Controllable semantic parsing via retrieval augmentation. In *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 7683–7698. Association for Computational Linguistics.

Patti J. Price. 1990. Evaluation of Spoken Language Systems: the ATIS Domain. In *HLT*.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Pawel Krzysztof Nowak, Tal Linzen, Fei Sha, and Kristina Toutanova. 2022. Improving compositional generalization with latent structure and data augmentation. In *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 4341–4362. Association for Computational Linguistics.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. *CoRR*, abs/2204.00498.

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, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 922–938. Association for Computational Linguistics.

Zejiang Shen, Hunter Lang, Bailin Wang, Yoon Kim, and David A. Sontag. 2024. Learning to decode collaboratively with multiple language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 12974–12990. Association for Computational Linguistics.

Sam Spilsbury, Pekka Marttinen, and Alexander Ilin. 2024. Generating demonstrations for in-context compositional generalization in grounded language learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15960–15991, Miami, Florida, USA. Association for Computational Linguistics.

Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A. Saurous, and Yoon Kim. 2023. Grammar prompting for domain-specific language generation with large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Yanzheng Xiang, Qian-Wen Zhang, Xu Zhang, Zejie Liu, Yunbo Cao, and Deyu Zhou. 2023. G^3r: A graph-guided generate-and-rerank framework for complex and cross-domain text-to-sql generation. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, volume ACL 2023 of *Findings of ACL*, pages 338–352. Association for Computational Linguistics.

Jingfeng Yang, Haoming Jiang, Qingyu Yin, Danqing Zhang, Bing Yin, and Diyi Yang. 2022a. SEQZERO: Few-shot compositional semantic parsing with sequential prompts and zero-shot models. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 49–60, Seattle, United States. Association for Computational Linguistics.

Jingfeng Yang, Le Zhang, and Diyi Yang. 2022b. SUBS: subtree substitution for compositional semantic parsing. In *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 169–174. Association for Computational Linguistics.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Minkai Xu, Joseph E. Gonzalez, Bin Cui, and Shuicheng Yan. 2024. SuperCorrect: Advancing Small LLM Reasoning with Thought Template Distillation and Self-Correction. In *International Conference on Learning Representations*.

Yuekun Yao and Alexander Koller. 2024. Simple and effective data augmentation for compositional generalization. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 434–449. Association for Computational Linguistics.

Jiacheng Ye, Zhiyong Wu, Jiangtao Feng, Tao Yu, and Lingpeng Kong. 2023. Compositional exemplars for in-context learning. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings*

of Machine Learning Research, pages 39818–39833. PMLR.

Pengcheng Yin, Hao Fang, Graham Neubig, Adam Pauls, Emmanouil Antonios Platanios, Yu Su, Sam Thomson, and Jacob Andreas. 2021. Compositional generalization for neural semantic parsing via span-level supervised attention. In *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 2810–2823. Association for Computational Linguistics.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *AAAI/IAAI, Vol. 2*.

Yury Zemlyanskiy, Michiel de Jong, Joshua Ainslie, Panupong Pasupat, Peter Shaw, Linlu Qiu, Sumit Sanghai, and Fei Sha. 2022. Generate-and-retrieve: use your predictions to improve retrieval for semantic parsing. pages 4946–4951.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2023. Automatic chain of thought prompting in large language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Shitian Zhao, Renrui Zhang, Xu Luo, Yan Wang, Shanghang Zhang, and Peng Gao. 2024. Unleashing the potentials of likelihood composition for multi-modal language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, volume EMNLP 2024 of *Findings of ACL*, pages 10152–10163. Association for Computational Linguistics.

Jing Zheng, Jyh-Herng Chow, Zhongnan Shen, and Peng Xu. 2023. Grammar-based decoding for improved compositional generalization in semantic parsing. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 1399–1418, Toronto, Canada. Association for Computational Linguistics.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Chunlin Zhu, Yuming Lin, Yaojun Cai, and You Li. 2026. Scot2s: Self-correcting text-to-sql parsing by leveraging llms. *Comput. Speech Lang.*, 95:101865.

A Dataset Statistics

Dataset	Spilt	Train	Fewshot	Validation	Test
SMCalFlow_{0c}	0c	25358	0	1324	663
SMCalFlow_{8c}	8c	25358	8	1324	663
SMCalFlow_{16c}	16c	25358	16	1324	663
SMCalFlow_{32c}	32c	25358	32	1324	663
$\text{GeoQuery}_{32}^{\text{TMPL}}$	`template`	441	32	0	407
$\text{GeoQuery}_{32}^{\text{TMCD}}$	`tmcd`	440	32	0	408
$\text{GeoQuery}_{32}^{\text{TMPL}}$	`template`	441	32	0	407
$\text{COGS}_{36}^{\text{vf}}$	`gen`	24000	36	0	1050
$\text{COGS}_{36}^{\text{vf}}$	`iid`	24000	36	0	1000
ATIS_{32}	`template0`	405	32	0	535
CFQ_{32}	`MCD1`	95743	32	1000	1000
CUDON_{32}	`tmcd.12088598`	154586	32	0	1000
mTOP_{32}	–	1000	32	1000	1000

Table 8: Dataset train, fewshot, validation and test statistics

B Hyperparameters

Hyperparameter	Value
Batch Size	128 (using gradient accumulation if large batches could not fit into memory)
Training Epochs	100
Optimizer	`AdamW` (using optimizer defaults if not specified)
Learning Rate	10^{-4}
Learning Rate Warmup	0.04
Entropy Bonus	0.1
Weight Decay	0.01
Gradient Clip	1.0
Seed Value	0
Precision	`bf16`
Batch Balance Ratio	0.5

Table 9: Hyperparameters used for training SLM

C Prompt Examples

We show schematic examples of the prompts used for each method evaluated in this paper. All prompts share a common *correction prompt* wrapper (Section C.1) that frames the task. The methods differ in how $\mathcal{S}$ (the error/instruction context) is constructed and whether the SLM's initial guess is included.

C.1 Correction Prompt Wrapper

All methods use the following outer structure at inference time. The `[S]` block varies by method and is described in subsequent sections. The `PROPOSED_OUTPUT` line is included when the SLM guess is provided ("+ SLM $\hat{y}$" variants in Table 3).

```
You are an expert semantic parser and editor. Given a few reference input-
output examples, return the OUTPUT for a given INPUT.

[S]

### Reference Examples (always correct) ###
```

```
INPUT: <retrieved or few-shot example 1>
OUTPUT: <gold output 1>
INPUT: <retrieved or few-shot example 2>
OUTPUT: <gold output 2>
...

### Instructions ###
1. Consider carefully if edits are necessary. Only correct errors matching a known error pattern.
2. Correct only what is necessary.
3. Follow the formatting and style of the examples. Do NOT invent unsupported forms.
4. Do NOT change ordering of existing elements. Only add or delete elements.
5. Propose edits step by step in the reasoning, specifying which known error
   case each fix corresponds to.

### Current Problem Instance ###
INPUT:
<test input>

PROPOSED_OUTPUT (small model):
<SLM prediction for test input>
```

C.2 EoT (Error-of-Thought)

The $\mathcal{S}$ block for EoT is constructed in two phases. First, the LLM generates concise GENERAL ERROR INSTRUCTIONS by analyzing SLM predictions on $\mathcal{D}_{\text{comp}}$. Then, for each few-shot example, the LLM generates per-example ERROR REASONING conditioned on the general instructions. The final $\mathcal{S}$ interleaves these with the SLM predictions and gold outputs:

```
## ERROR-CORRECTION INSTRUCTIONS
Use the GENERAL ERROR INSTRUCTIONS, plus the interleaved training examples.

## GENERAL ERROR INSTRUCTIONS
<LLM-generated checklist, e.g.:>
- If input mentions "X's manager", ensure FindManager wraps the person reference.
- If BACKTRANSLATED_INPUT matches INPUT, do NOT propose edits.
- ...

### EXAMPLE
INPUT:
schedule a meeting with Tom's team

MODEL OUTPUT:
CreateEvent(with("Tom's team"),...)

BACKTRANSLATED INPUT (from MODEL OUTPUT):
schedule a meeting with "Tom's team"

ERROR REASONING:
<LLM-generated analysis, e.g.:>
- Input says "Tom's team" (possessive), requires FindTeamOf(Tom), not literal.
- BACKTRANSLATED_INPUT treats it as literal string, confirming the error.
- Fix: replace with("Tom's team") with with(FindTeamOf("Tom"))

TRUE OUTPUT:
CreateEvent(with(FindTeamOf("Tom")),...)

### EXAMPLE
INPUT:
create a lunch meeting on Friday

MODEL OUTPUT:
CreateEvent(time(Friday),subject("lunch"))

BACKTRANSLATED INPUT (from MODEL OUTPUT):
create a lunch meeting on Friday

ERROR REASONING:
- MODEL OUTPUT matches TRUE OUTPUT.
- BACKTRANSLATED_INPUT matches INPUT.
- NO ERRORS detected.
```

```
TRUE OUTPUT:
CreateEvent(time(Friday),subject("lunch"))

...
```

C.3 ERP (Error Reflection Prompting)

ERP follows the same structure as EoT, but the error analysis is based on *LLM predictions* (leave-one-out) rather than SLM predictions. There is no backtranslation step, and the examples show `LLM OUTPUT` instead of `MODEL OUTPUT`:

```
## ERROR-REFLECTION INSTRUCTIONS
(from LLM-generated examples)

## GENERAL ERROR INSTRUCTIONS
<LLM-generated checklist from LLM errors>

### EXAMPLE
INPUT:
schedule a meeting with Tom's team

LLM OUTPUT:
CreateEvent(with(FindTeamOf("Tom")),duration(hours(1)))

ERROR REASONING:
- LLM added spurious duration(hours(1)) not present in input.
- Fix: remove the duration constraint.

TRUE OUTPUT:
CreateEvent(with(FindTeamOf("Tom")),...)

...
```

C.4 Error Examples (Error Ex.)

In this variant, erroneous SLM predictions are shown alongside the gold outputs, but *without* any LLM-generated error analysis:

```
### Known Errors
The following are examples of erroneous
predictions made by the model.

INPUT: schedule a meeting with Tom's team
MODEL OUTPUT:
  CreateEvent(with("Tom's team"),...)
BACKTRANSLATION FROM MODEL OUTPUT:
  schedule a meeting with "Tom's team"
TRUE OUTPUT:
  CreateEvent(with(FindTeamOf("Tom")),...)

INPUT: add a call with Ana and her manager
MODEL OUTPUT:
  CreateEvent(with("Ana","manager"),...)
BACKTRANSLATION FROM MODEL OUTPUT:
  add a call with Ana and manager
TRUE OUTPUT:
  CreateEvent(with("Ana",FindManager("Ana")),...)

...
```

C.5 Directions (Only Directions / Directions + SLM $\hat{y}$)

The LLM is asked to infer transformation rules from the few-shot pairs. The $\mathcal{S}$ block contains these rules instead of error examples:

```
## SUMMARY OF RULES TO FOLLOW
<LLM-generated rules, e.g.:>
1. Top-level structure: all outputs use CreateEvent(...) wrapper.
2. Attendees: possessive references (X's manager) -> FindManager(X).
```

```
3. Teams: "X's team" -> FindTeamOf(X).
4. Dates: "tomorrow" -> Tomorrow(), "this week" -> ThisWeek().
5. Subject: quoted string in subject(...).
...
```

C.6 Grammar Prompt

Following Wang et al. (2023), we induce a BNF-style grammar from the few-shot data, then extract per-example grammar subsets:

```
## GRAMMAR INSTRUCTIONS
You are an expert programmer. Write a program for the given natural language query.
The following are a general inferred grammar and per-example grammars.

GENERAL GRAMMAR:
<event> ::= CreateEvent(<args>)
<args>  ::= <attendee> | <time> | ...
<attendee> ::= with(<person>)
<person> ::= "name" | FindManager(<person>)
           | FindTeamOf("name")
...

EXAMPLE INPUT:
schedule a meeting with Tom's team
EXAMPLE GRAMMAR SUBSET:
<event> ::= CreateEvent(<attendee>,...)
<attendee> ::= with(FindTeamOf("name"))
EXAMPLE OUTPUT:
CreateEvent(with(FindTeamOf("Tom")),...)

...
```

C.7 SetBSR / GPT-5 (Baselines)

For the RAG and few-shot baselines, $\mathcal{S}$ is empty. The prompt contains only the reference examples (retrieved via SetBSR or provided as few-shots) and the test input. No SLM guess is provided:

```
You are an expert semantic parser and editor. Given a few reference input-output examples, return the OUTPUT for a given INP

### Reference Examples (always correct) ###
INPUT: <retrieved example 1>
OUTPUT: <gold output 1>
...

### Instructions ###
1. Follow the formatting and style of the examples.
2. For each component, check the examples and think carefully about where it should appear in the OUTPUT.
3. Propose the parts step by step the reasoning, then provide the OUTPUT.

### Current Problem Instance ###
INPUT:
<test input>
```

D Examples of Failure Cases

We present representative examples from each failure category identified in Table 7. In each example, **SLM** is the T5 prediction, **EoT** is the final corrected output and **Gold** is the ground truth. Erroneous portions are <u>underlined</u>.

D.1 Missed Correction

EoT fails to fix a non-compositional SLM error. The SLM makes a parsing mistake that would also occur on in-distribution data, and EoT does not correct it.

GeoQuery. *"what is the population of the smallest state"*

```
Gold: answer(smallest(population_1( state(all))))
SLM:  answer(population_1(smallest(state(all))))
EoT:  answer(population_1(smallest(state(all))))
```

The correct nesting requires `smallest` to wrap `population_1`, selecting the state with the smallest population. Both the SLM and EoT place `population_1` outside `smallest`, which instead computes the population of the smallest state (by area). EoT fails to detect the operator ordering error.

D.2 Missed Composition

EoT fails to fix a compositional generalization error. The SLM misses a compositional pattern and EoT does not fully correct it.

COGS. *"Ryan sent the frog Paula ."*

```
Gold: send(agent=Ryan, recipient=*frog, theme=Paula)
SLM:  send(agent=Ryan, recipient=Paula, theme=*frog)
EoT:  send(agent=Ryan, recipient=Paula, theme=*frog)
```

The ditransitive "sent the frog Paula" has recipient=`*frog` and theme=`Paula`. The SLM reverses these roles and EoT does not correct the swap.

D.3 Overcorrect Add

EoT introduces a structural element not present in the gold output, overriding a correct or partially correct SLM prediction.

GeoQuery. *"how big is texas"*

```
Gold: answer(size(stateid(texas)))
SLM:  answer(size(stateid(texas)))
EoT:  answer(size(state(stateid(texas))))
```

The SLM output was already correct. EoT unnecessarily wrapped `stateid(texas)` with an extra `state(...)` predicate, a hallucinated structural element not supported by the gold form.

SMCalFlow. *"when can fix a meeting with my staff"*

The SLM correctly used `FindReports` (the user's direct reports). EoT replaced it with `FindTeamOf`, a different semantic function that refers to a named person's team rather than the current user's reports—a mistaken semantic substitution.

D.4 Overcorrect Remove

EoT removes a correct element from the SLM output during correction.

GeoQuery. *"what state is austin the capital of"*

```
Gold: answer(state(loc_1(capital(cityid(austin,_)))))
SLM:  answer(state(loc_1(capital(cityid(austin,_)))))
EoT:  answer(state(capital(cityid(austin,_))))
```

The SLM output was already correct. EoT removed the required `loc_1` wrapper around `capital`, breaking the location resolution logic.

D.5 Overcorrect Reorder

EoT produces an output with the correct elements but in the wrong order.

COGS. *"Olivia forwarded a father Paula ."*

```
Gold: forward(agent=Olivia, recipient=father, theme=Paula)
SLM:  forward(agent=Olivia, recipient=father, theme=father)
EoT:  forward(agent=Olivia, recipient=Paula, theme=father)
```

The SLM duplicated `father` in the theme slot. EoT attempted to fix this by inserting `Paula`, but placed it in the recipient slot instead of the theme slot, swapping the two roles.

D.6 Formatting

The EoT output contains all the correct semantic elements but has a syntax error such as mismatched parentheses or missing delimiters.

COGS. *"Emily gave Emma the donut in a basin beside a tree in a house on a table beside the chair beside the stage on the boat ."*

```
Gold: give(agent=Emily, recipient=Emma,
  theme=*donut(nmod.in=basin(
    nmod.beside=tree(nmod.in=house(
      nmod.on=table(nmod.beside=*chair(
        nmod.beside=*stage(
          nmod.on=*boat))))))))
EoT:  give(agent=Emily, recipient=Emma,
  theme=*donut(nmod.in=basin(
    nmod.beside=tree(nmod.in=house(
      nmod.on=table(nmod.beside=*chair(
        nmod.beside=*stage(
          nmod.on=*boat)))))))
```

The EoT output is semantically correct but is missing one closing parenthesis. The SLM output had severe structural errors (repeated `nmod.on=*stage` nesting). EoT reconstructed the correct deep nesting structure but miscounted the parentheses at the end.

SMCalFlow. *"I need a meeting with Abby and her skip"*

The EoT output correctly composed the double `FindManager(FindManager(...))` for "skip" (skip-level manager), but dropped one closing parenthesis from a 588-character output, producing an invalid expression. This is a common failure mode for long outputs with deep nesting.

E SLM Errors Corrected by LLM

We present examples from each dataset where the fine-tuned SLM (T5) produces an incorrect output but the LLM with SetBSR retrieval produces the correct one. These correspond to the $\times_{\text{SLM}}\checkmark_{\text{LLM}}$ cell in Table 1. For each example we show the input, what the SLM got wrong, and the correct output that the LLM produced.

E.1 SMCalFlow

A recurring SLM failure on SMCalFlow is treating compositional references (e.g., "my team", "my boss") as literal strings in the `:subject` field instead of resolving them with the appropriate `Find*` function call.

Example 1. *"I have to see my team on Monday"*

```
SLM: CreateEvent(
       start(Monday),
       subject("see my team"))

LLM: CreateEvent(
       attendees(FindTeamOf(CurrentUser())),
       start(Monday))
```

The SLM places "see my team" as a literal subject string. The LLM correctly interprets "my team" as requiring a `FindTeamOf` call on the current user and creates an attendee constraint instead.

Example 2. *"i have to dinner my team members"*

```
SLM: CreateEvent(
       subject("dinner my team members"))

LLM: CreateEvent(
       attendees(FindTeamOf(CurrentUser())),
       subject("dinner"))
```

The SLM concatenates the entire phrase into the subject. The LLM separates "dinner" as the subject from "my team members" as the attendee constraint, correctly using `FindTeamOf`.

E.2 GeoQuery

SLM failures on GeoQuery often involve missing wrapper predicates or using incorrect predicate names.

Example 1. *"how large is texas"*

```
SLM: answer(size(texas))
LLM: answer(size(stateid(texas)))
```

The SLM omits the required `stateid` wrapper that identifies "texas" as a state entity. The LLM correctly includes it.

Example 2. *"how tall is mount mckinley"*

```
SLM: answer(tall(loc_1(
       placeid(mount mckinley))))
LLM: answer(elevation_1(
       placeid(mount mckinley)))
```

The SLM uses a non-existent predicate `tall` with an unnecessary `loc_1` wrapper. The LLM maps "how tall" to the correct domain predicate `elevation_1`.

Example 3. *"which states have a river"*

```
SLM: answer(state(loc_1(river(
       riverid(riverid(riverid(...)))))))
LLM: answer(state(loc_1(river(all))))
```

The SLM enters a degenerate loop, nesting `riverid` repeatedly. The LLM correctly uses `river(all)` to quantify over all rivers.

E.3 COGS

SLM failures on COGS fall into two main patterns: ditransitive argument confusion and structural collapse on deeply nested sentences.

Example 1: Ditransitive argument confusion. *"The child gave a dog Paula ."*

```
SLM: give(agent=*child,
      recipient=dog, theme=dog)
LLM: give(agent=*child,
      recipient=dog, theme=Paula)
```

The SLM duplicates "dog" in both the recipient and theme slots, failing to assign "Paula" as the theme. The LLM correctly identifies the three distinct arguments of the ditransitive verb.

Example 2: Nested clauses with modifiers. *"A boy on a table noticed that a visitor in a castle slept ."*

```
SLM: on(agent=boy(nmod.on=table),
      ccomp=sleep(agent=visitor))

LLM: notice(agent=boy(nmod.on=table),
      ccomp=sleep(
        agent=visitor(nmod.in=castle)))
```

The SLM fails catastrophically: it uses `on` as the main predicate instead of `notice` and drops the locative modifier `nmod.in=castle` from "visitor". The LLM correctly identifies the main verb, preserves both nominal modifiers, and nests the complement clause.

Example 3: Embedded compositional clause. *"A cat liked that Matthew passed Olivia Paula ."*

```
SLM: like(agent=cat,
      ccomp=pass(agent=Matthew,
        theme=Olivia))

LLM: like(agent=cat,
      ccomp=pass(agent=Matthew,
        recipient=Olivia, theme=Paula))
```

The SLM drops the third argument "Paula" entirely from the embedded ditransitive clause. The LLM correctly produces the full three-argument structure within the `ccomp` clause.

ERP	EoT
INPUT: Make a meeting with Abby and her team ... ERROR REASONING Two attendee constraints are emitted separately instead of being combined under a single andConstraint; attendees must be grouped. The FindTeamOf result is incorrectly wrapped in AttendeeListHasRecipient; it should be wrapped in AttendeeListHasPeople. The named Abby recipient usage is otherwise correct and should remain the recipient argument to FindTeamOf (do not convert the explicit name into a different find call). Fix: replace the second AttendeeListHasRecipient(FindTeamOf()) with AttendeeListHasPeople(FindTeamOf()) and place both attendee nodes inside one andConstraint.	INPUT: Make a meeting with Abby and her team ... ERROR REASONING: The model omitted the team constraint: original input asked for "Abby and her team" but backtranslated output only includes Abby, so AttendeeListHasPeople(FindTeamOf(recipient(Abby))) is missing. There is no andConstraint combining Abby and her team in the model output; attendees only list Abby as a recipient. No existing andConstraint or manager/team wrapper is present, so adding the team constraint will not cause overcorrection. Fix: replace the single AttendeeListHasRecipient entry with an andConstraint that contains both AttendeeListHasRecipient for Abby and AttendeeListHasPeople with FindTeamOf for the same Abby recipient. Ensure the FindTeamOf uses the same RecipientWithNameLike constraint for Abby so both attendee constraints refer to the identical recipient entity.

Table 10: Side-by-side comparison of Error Reflection Prompts and Error-of-Thought Prompts on SMCalFlow32

F Error Reflection Prompts vs Error-of-Thought Prompts

In Table 10 we compare an indicative case of error-reasoning mismatch for the SLM errors from ERP errors. Since ERP errors are induced by using retrieved examples to solve the fewshots, the errors can be different. In this case, the few-shot prompted LLM was able to make the compositional generalization work correctly, but had errors such as using `andConstraint` incorrectly and using `AttendeeListHasRecipient` instead `AttendeeListHasPeople`. On the other hand, the SLM wrote “Abby and her team" literally and this is flagged for correction in the EoT reasoning trace.

G Details about dataset setups

On **SMCalFlow**, we make a similar adjustment as in (Qiu et al., 2022) where we exclude all examples from training data where the output logical form contained literal quotes to something that didn’t exist in the input string. In total that is 45 examples. These sorts of examples are misleading to the model since quoted regions always indicate something to be copied.

On **GeoQuery** we substituted back in the place names after making the dataset partition in Finegan-Dollak et al. (2018). The original authors of the compositional splits “anonymized" the data for the reason that the place names were irrelevant to the compositional problem. In our view, this results in many duplicate entries that reduces how interesting the problem is. We substituted those place names back.

On **mTOP** we sampled only 1000 training examples, similar to the evaluation case in Zemlyanskiy et al. (2022). These examples were sampled at random from the training dataset using the same seed each time the training was run. mTOP includes three different languages – English, Italian and Japanese. We sampled with equal likelihood from all three according to the original dataset balance.

On **GeoQuery**, **COGS**, **CUDON**, **CFQ** and **mTOP** we needed to construct a few-shot set based on examples in the compositional generalization test set, since there was no separately provided fewshot set. To do this, we shuffle the test data based on a fixed random seed, then take the first 32 examples as the few-shot data and keep the rest as test data. For **COGS**, there are annotations of each compositional generalization case along with the data. This isn't included in the "vf" version of the dataset, but it can be obtained from the orignal version and joining the two keyed on the input string. We use these annotations both to obtain few-shot examples and to select the test set - we select 3 examples from each generalization case to from part of the few-shot data and 50 examples from each case to form the test data. The overall test set size is therefore 1050 examples.

H Details about baseline implementations

We reimplemented all baselines from scratch, based a best-effort basis from the detailed disclosed in the SetBSR (Gupta et al., 2023), GandR (Zemlyanskiy et al., 2022), CASPER (Pasupat et al., 2021), CoT (Zhang et al., 2023; Wei et al., 2022) and ERP papers (Li et al., 2025a). We used LLM code generation tools to help in the reimplementation, and carefully checked and debugged the resulting code ourselves for correctness and intended output.

For all retrieval methods, we set up a baseline retriever by encoding sentence inputs using the `all-MiniLM-L6-V2` model from the "sentence-transformers" package and adding those vectors into a FAISS vector database. The vectors are standard-scaled and transformed with PCA such that they are projected on their 256 highest variance principal components, then normalized. The index itself is just a simple inner-product index using cosine similarity as the metric.

For **SetBSR**, our implementation works by first retrieving 16 neighbors using the index, then pruning that list with the SetBSR algorithm. The algorithm computes all-pairs token-embedding score using a SentenceBERT model between the query and each retrieved example. Then it prunes retrieved examples by picking only the subset which maximize so-called set-coverage across all the different tokens in the input. Redundant examples are dropped.

For **GandR**, our implementation first generates a guess $\hat{y}$ using the SLM as usual. The FAISS index in this case is based on the combined input and output text. We concatenate the query input to the guessed output from the SLM and use that to search the index. 16 neighbors are retrieved using this method.

For **CASPER**, our implementation uses SetBSR to retrieve 16 neighbors first, then there is a prompt to an LLM (in this case GPT-5-mini) asking to generate 16 new neighbors based on the test-time input x_q and the fetched neighbors. Then these synthetic neighbors are used as the nearest neighbors for final LLM inference of x_q.

Dataset	SLM	LLM	SLM✓ LLM✓	SLM✓ LLM✗	SLM✗ LLM✓	SLM✗ LLM✗	Disagree	Union
SMC_{32}	65.16	59.28	47.06	18.10	12.22	22.62	30.32	77.38
GQ_{32}^{TMPL}	83.05	73.46	65.85	17.20	7.62	9.34	24.82	90.66
GQ_{32}^{TMCD}	63.73	77.45	55.64	8.09	21.81	14.46	29.90	85.54
$COGS_{36}^{vf}$	85.14	82.76	71.14	14.00	11.62	3.24	25.62	96.76
CFQ_{32}	66.81	21.09	16.43	50.39	4.66	28.52	55.05	71.48
$ATIS_{32}$	63.56	55.07	44.93	18.63	10.14	26.29	28.78	73.71
$CUDON_{32}$	52.50	66.50	40.20	12.30	26.30	21.20	38.60	78.80
mT_{32}	25.73	40.13	20.27	5.47	19.87	54.40	25.33	45.60

Table 11: Confusion matrix (%) between fine-tuned T5 (SLM) and GPT-5 with SetBSR retrieval (LLM) across all evaluated datasets. **SLM** and **LLM** columns show overall accuracy. The four central columns show the percentages of joint outcomes. **Disagree** is the sum of the two off-diagonal cells. **Union** is the upper bound when always selecting the correct answer from either method.

In **CoT** we use the AutoCoT methodology in (Zhang et al., 2023) as opposed to manually specifying the chains of thought ourselves. There is a preprocessing step where each pair in the few-shot data is given to the LLM, along with instructions to produce instruction for how to construct the output from

the input. A similar method is used for **ERP**, except that we provide the generated output from an LLM which was given the 5 nearest neighbours for each few-shot example as part of its prompt. For **EoT**, we use the SLM initial guess instead.

I Details about LLM setups

The specific checkpoints used for the smaller LLMs were:

- **GLM**: `zai/GLM-4.7-flash`
- **Gemma 3**: `google/gemma-3-27b-it`
- **Qwen**: `qwen/qwen3.5-35b-a3b`

We use the 16 bit precision checkpoints, but do not apply any inference-time quantization.

As explained in Section 3.12, it was difficult to get evaluation on the smaller sized LLMs to work in the way that we liked. When evaluating on GPT-5-mini, we always forced JSON responses with objects where the first field was “reasoning" and the next field was “output". GPT can follow this structure well, and placing the “reasoning" field first provides a useful reminder that GPT should always provide some reasoning before the final answer, but that the two should be separately parseable. Unfortunately, this did not work very well on the smaller LLMs, where constrained decoding would more often than not interfere with the responses we were trying to generate. Therefore, we modified the instructions and loosened the task such that the LLM could do whatever reasoning it wanted to on earlier lines, but would provide the final answer on the last line.

We found that the smaller LLMs were not able to follow the details of the answer formatting requirements, which initially reduced exact match scores significantly. This is because the answer format encodes the tokenization scheme – spaces are placed between parentheses and quotation marks for example. To handle this, we apply a simple post-hoc regex replacement to the final answers, defined on a per-dataset basis, applied equally to all methods used on each dataset.

We also needed to disable the thinking mode on models with thinking support using the `</think>` tag. If thinking or any kind of long-form reasoning was enabled, we found that these models would get stuck in “thinking loops", particularly where they tried to self-discover edge cases and possible trick questions by recalling various irrelevant examples in the prompt.

This approach mostly worked, except when evaluating the SetBSR case on LoRA fine-tuned models. In this case, we needed to disable thinking and reasoning entirely, by removing any requests to do step-by-step reasoning from the prompt and asking the model to only output the answer.

When fine-tuning the models with LoRA. we wanted to ensure that we might teach the LLM about the specific syntax and idea behind each of the datasets without destroying the built-in reasoning and instruction following-capabilities by overtraining. Therefore, we fine-tuned with LoRA on quite conservative settings. Fine-tuning was only done for one training epoch and the LoRA values were more or less the defaults: a LoRA rank of 8, alpha of 16 and dropout of 0.05, targeting the query and value projection matrices of the attention operation. For GLM we target the output projection matrix.

J Full confusion matrix for SLM vs LLM over all datasets

We extend the confusion matrix analysis from Table 1 to all evaluated datasets. Table 11 shows the full 2×2 contingency table between the fine-tuned SLM (T5) and LLM with SetBSR retrieval (GPT-5 mini) for each dataset. The *Disagree* column is the total percentage of cases where exactly one method is correct. The *Oracle* column shows the upper bound achievable by always selecting the correct answer when either method succeeds.

Several patterns emerge across the full set of datasets. First, the disagreement rate is consistently high, ranging from 25% (mTOP, COGS) to 55% (CFQ), confirming that the SLM and LLM make substantially different errors regardless of the dataset.

Second, the balance between SLM-exclusive and LLM-exclusive correct answers varies by dataset. On CFQ, the SLM dominates (50.4% SLM-only vs. 4.7% LLM-only), consistent with the LLM’s poor overall

	SLM → EoT		SLM → RAG	
	✗ → ✓	✓ → ✗	✗ → ✓	✓ → ✗
SMC_{32}	38.90	1.12	21.11	4.07
GQ_{32}^{TMPL}	46.03	22.67	11.30	12.28
$COGS_{32}^{vf1k}$	79.58	3.17	15.84	4.65
CFQ_{32}	40.68	4.87	14.05	75.42
$CUDON_{32}$	54.32	11.81	55.37	23.43
mT_{32}^{1000}	20.55	23.79	26.75	21.24
$ATIS_{32}$	24.57	16.88	27.84	29.32
GQ_{32}^{TMCD}	52.03	6.15	60.14	12.69

Table 12: Conditional error analysis of correcting SLM responses with EoT versus replacing SLM with LLM-RAG (SetBSR) responses across all datasets. Numbers are percentages of the original set size (eg, percentage of wrong answers or percentage of right answers). Note that the set sizes can be different, so a larger percent value in ✗ → ✓ versus ✓ → ✗ does not necessarily indicate that the corrected more instances than were overcorrected, but rather that the share was bigger. The first three rows reproduce Table 4; subsequent rows extend to remaining datasets.

performance on SPARQL generation. Conversely, on CUDON and GeoQuery-TMCD, the LLM has a larger share of exclusive correct answers (26.3% and 21.8%), reflecting the benefit of retrieval-augmented generation on these datasets.

Third, the union upper bound is substantially higher than either individual method in all cases: on average 14.6 percentage points above the better of the two methods. This reinforces the motivation for combining both approaches through error correction, as pursued in EoT.

Finally, the datasets where EoT shows the largest gains in Table 2 (SMCalFlow, COGS, CFQ, CUDON) tend to be those with both a high disagreement rate and a high oracle ceiling, suggesting that EoT is most effective when the two models have complementary strengths and there is room for improvement.

K Conditional Error Analysis for All Datasets

We extend the conditional error analysis from Table 4 to the remaining datasets. Table 12 shows the rate at which EoT corrects SLM errors (✗ → ✓) and the rate at which it introduces new errors on previously correct items (✓ → ✗), compared with the same when the SLM response is replaced by SetBSR.

The results for the new datasets reinforce the patterns observed in Table 4. On CFQ, CUDON, and GeoQuery-TMCD, EoT corrects a large share of SLM errors (40.7%, 54.3%, and 52.0% respectively) while introducing relatively few new errors on previously correct answers (4.9%, 11.8%, and 6.2%). This is the same favorable profile seen on SMCalFlow and COGS. On mTOP, EoT corrects 20.6% of errors but also degrades 23.8% of previously correct answers, yielding a net loss in accuracy consistent with mTOP's low oracle ceiling (45.6% in Table 11).

ATIS shows a mixed profile: EoT corrects 24.6% of SLM errors but degrades 16.9% of previously correct answers, resulting in a slight net decrease in accuracy (from 63.8% to 61.9%).

GeoQuery-TMCD shows a favorable EoT profile: 52.0% of SLM errors are corrected with only 6.2% degradation, yielding a net improvement of 15 percentage points. RAG corrects a slightly larger share of errors (60.1%) but at double the degradation rate (12.7%), making EoT the more conservative correction method on this dataset.

The SLM → RAG comparison reveals a clear contrast between EoT and retrieval-based correction. On CFQ, RAG degrades 75.4% of previously correct SLM answers while correcting only 14.1% of errors, confirming that SetBSR retrieval is poorly suited to SPARQL generation. In contrast, EoT corrects 40.7% of CFQ errors with only 4.9% degradation. On CUDON, RAG and EoT correct comparable shares of errors (55.4% vs. 54.3%) but RAG introduces twice the degradation (23.4% vs. 11.8%), making EoT the more reliable correction method. On mTOP, RAG corrects more errors than EoT (26.8% vs. 20.6%) with comparable degradation (21.2% vs. 23.8%). For ATIS, both methods degrade a substantial share of correct answers: RAG at 29.3% and EoT at 16.9%, confirming that ATIS is challenging for both correction approaches.