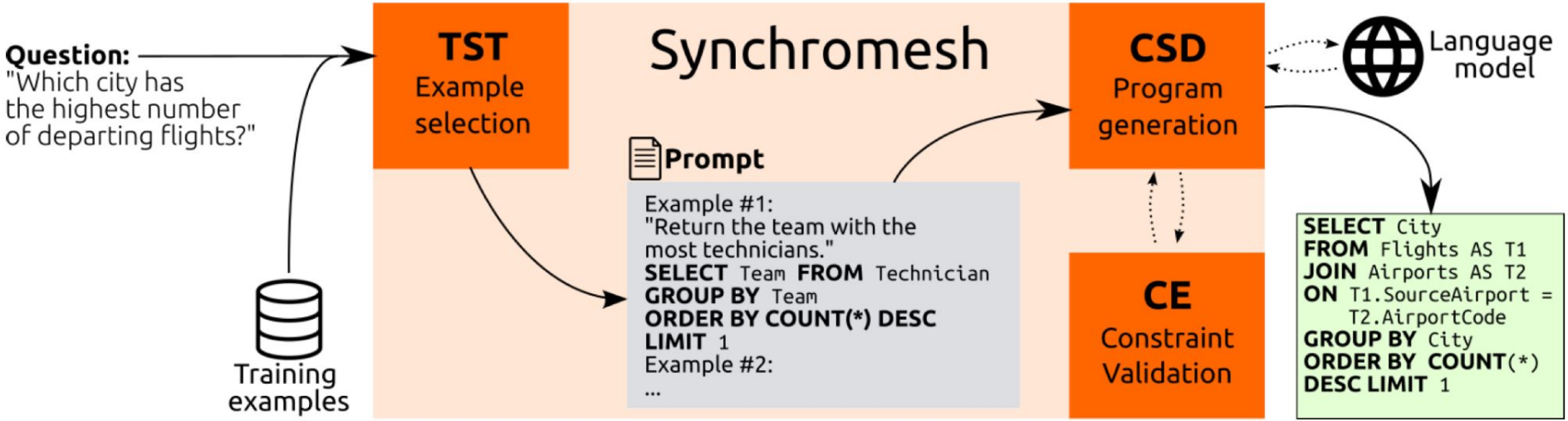
Sychromesh: Reliable code generation from pre-trained language models

Кульпин Павел
Аъзам Бехруз
Колесников Павел

Постановка задачи

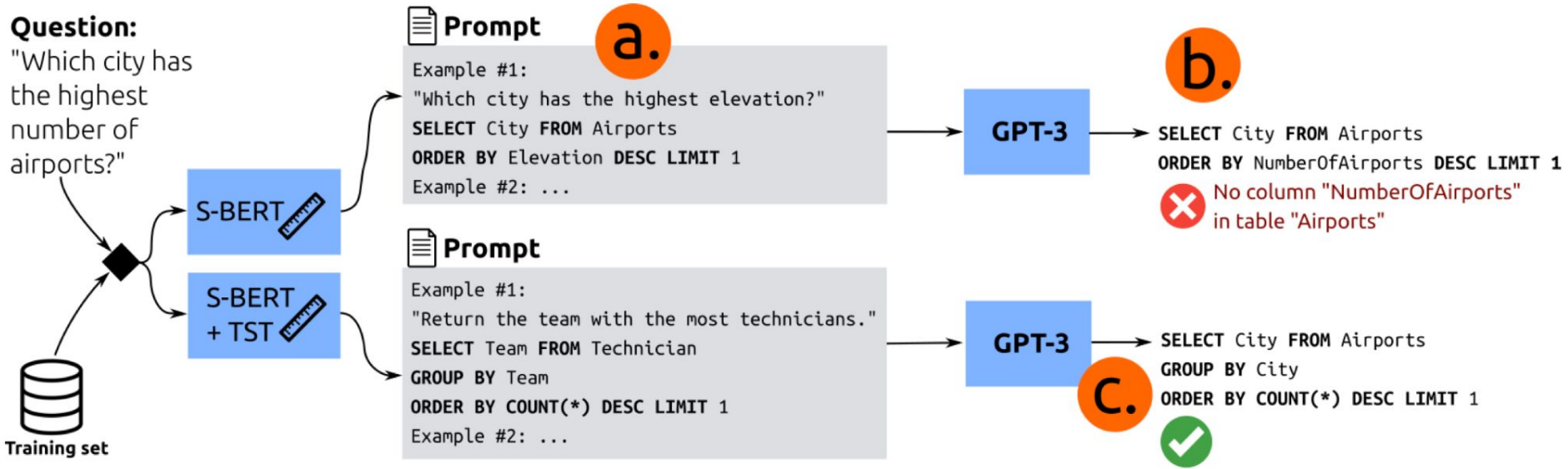


Нововведения

- **[TST]** Target Similarity Tuning
- **[CSD]** Constrained Semantic Decoding
- **[CE]** Completion engine

TST

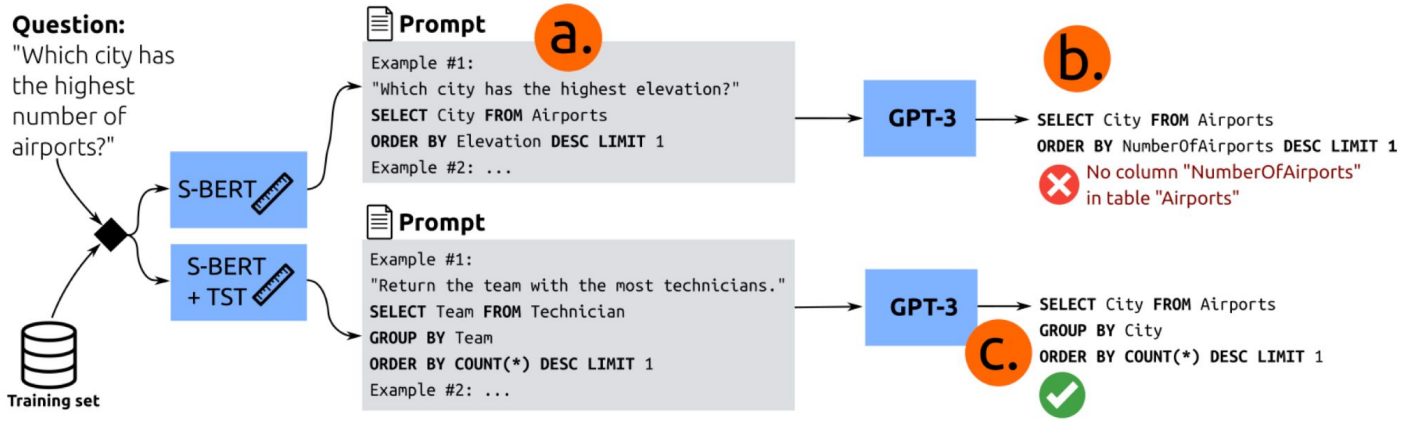
Выбирать релевантные подсказки не по текстовому описанию, а опираясь на структурную схожесть.



TST

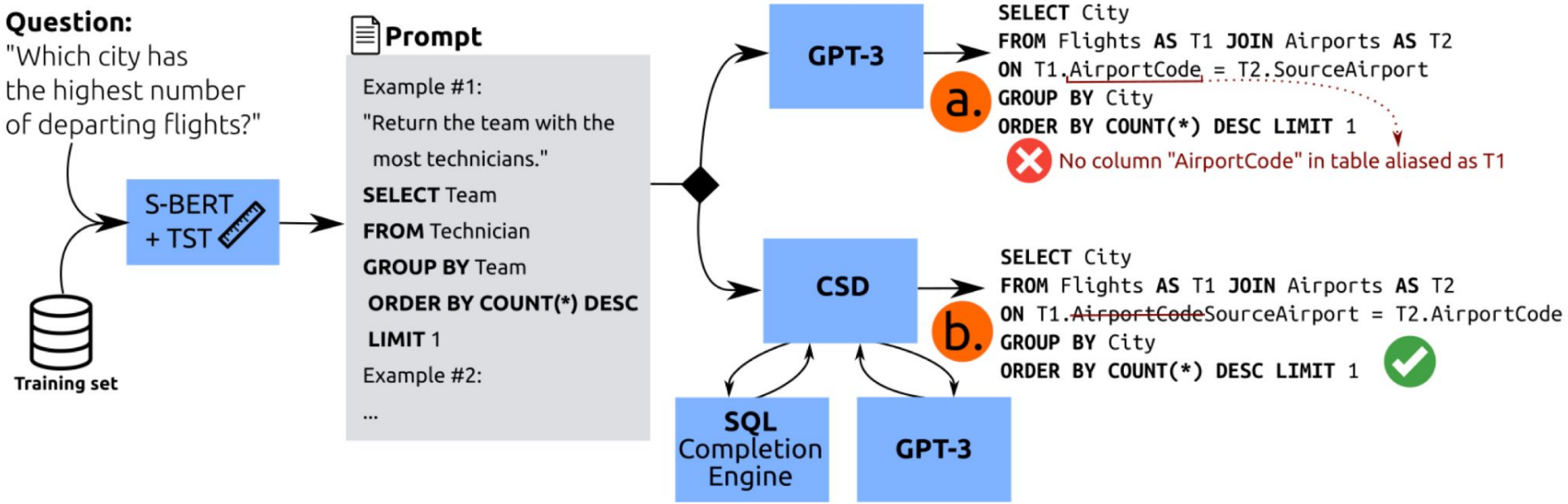
$$\mathcal{D}_i = (p_i, u_i) \qquad S(p_a, p_b) \in [0, 1]$$

$$\mathcal{L}_{TST}(\theta) := \mathbb{E}_{i,j \sim \mathcal{D}} \left[f_\theta(u_i, u_j) - S(p_i, p_j) \right]^2$$



CSD

- TST помогает иметь верную структуру, но этого недостаточно
-



CSD

- Весомая часть ошибочных выводов падают с ошибкой во время исполнения (RE).
- CSD исправляет ошибки путем построения кода, а не пост-фактум.
- CE - оракл, который для недописанной программы вернет набор всех токенов, дополняющих код до искомого.
- Модель генерирует код последовательно токен за токеном, а CSD гарантирует, что каждый i-й токен будет выбран из набора, возвращаемого CE.

CE

- **Σ** - базовый алфавит. **L** - целевой язык.
- **Σ_L ⊆ Σ_*** - набор валидных токенов целевого языка.
- => CE : **C_L** - частичная функция из **Σ_L** во множество токенов.

CE

Строки из **Dom(C_L)** - completion points и **C_L** удовлетворяет аксиомам:

1. Пустая строка и любой токен из **L** - являются completion points. Для любого **p ∈ L**: **C_L(p) = r`$`** - регулярное выражение, “соответствующее” какому-то токену.
2. Если **s** - completion point, и токен **t** соответствует **C_L(s)**, то **s*t** - также completion point.
3. И в обратную сторону: **s = t*t_0** - completion point и **t_0** - токен, то **t** - completion point и **C_L(t)** “подходит” **t_0.**

CE

Language	Constraint	Example of partial program	Valid/Invalid Examples
SQL	A valid identifier must follow after `AS`.	`SELECT Name, Role FROM User AS ^`	U ✓ T1 ✓ 2 ×
	Column names must come from schema, even behind aliases.	`SELECT U.Name FROM User AS U WHERE U. ^`	Name ✓ DoB ✓ Birthday ×
Vega-Lite	Data fields must be used with types compatible with their values.	`{"x": {"field": "Category", "type": ^`	`"nominal"` ✓ `"temporal"` ×
	Do not facet on field with too many distinct values (breaks rendering).	`{"column":{"field": ^`	`"Category"` ✓ `"ZipCode"` ×
SMCalFlow	Type-check parameters of all API functions.	`(Yield (PlaceHasFeature ( ^`	`Takeout` ✓ `IsWindy` × `List.Apply` ×
	Track declared variables and their types.	`(let (x 85) (Yield (inCelsius ^`	`x` ✓ `y` ×

Table 1: Examples of constraints implemented in our CEs for SQL, Vega-Lite and SMCalFlow. Given a partial program, CEs return a regular expression that matches the valid tokens that can follow. Here, we show positive and negative token examples for each such regular expression. This abstraction allows domain experts to encode a wide range of expressive code generation constraints.

Deriving completions for grammars

Парсеры языков генерируются из “грамматики” искомого языка, содержащей достаточно информации для построения контекстно-свободного слоя.

Авторы написали библиотеку, расширяющую парсеры, сгенерированные ANTLR:

1. Анализируется программный префикс.
2. Извлекаем его промежуточное состояние в Augmented Transition Network на текущем последнем токене.
3. Из этого состояния получаем все возможные токены-продолжения префикса.

Итого мы получаем список токенов-продолжений и их типов, а также частичное семантическое дерево префикса, которые подаются на вход CE.

CE - принятие решений

Пусть **L^c** - префикс-замыкание для языка **L**, то есть содержит все возможные программы языка и все их префиксы.

Начиная с пустого префикса **p**, мы ищем префикс **s** максимальной длины, который “подходит” **C_L(p)**, добавляем его к **p** и повторяем это действие пока токены не перестанут подходить.

Получаем 2 случая:

1. **p** - completion point (**∈ L^c)** и **s** - пустое.
2. **s** - остаток после удаления самого большого префикса во множестве completion points

Во втором случае нам важно, есть ли **c** - completion string - такая, что **sc** матчится с **C_L(p)**?

CE - принятие решений

1. **p** - completion point (**∈ L^c**) и **s** - пустое.
2. **s** - остаток после удаления самого большого префикса во множестве completion points

Для обоих случаев порядок действий определяется из производной Бжозовского: производной языка **L** по “направлению” строки **u** является множество {**v: uv ∈ S**} => эта производная определит, можно ли продолжить **s** до элемента из **C_L(p)**, сводя поиск к сопоставлению с регулярными выражениями **L^c**.

The Constrained Semantic Decoding Algorithm

- **Σ_M** - словарь модели, тогда **V_M** - множество следующих валидных токенов

$$V_M(s) = \{t \in \Sigma_M : st \in L^c\}$$

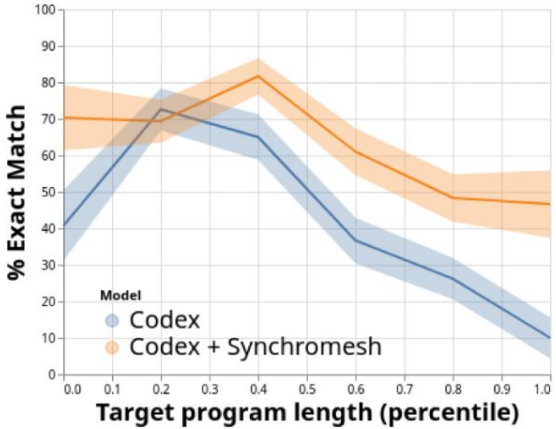
- Так мы гарантируем инвариантность последовательности генерируемых префиксов.
- Тонкость: разница токенизации. Хорошая новость: мы с ней не сталкиваемся!
- Вся процедура, по заявлению авторов, увеличивает сложность семплинга не более, чем на 8%.

Experiments

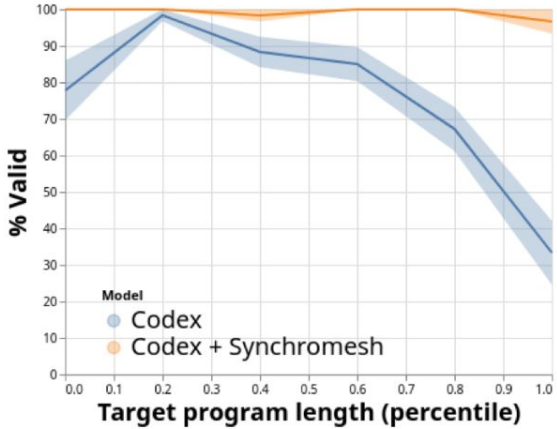
Model	SQL			Vega-Lite			SMCalFlow		
	Exec.	**Valid**	**Dist.**	**Acc.**	**Valid**	**Dist.**	**Acc.**	**Valid**	**Dist.**
Andreas et al. (2020)	-	-	-	-	-	-	72%$^{(S)}$	-	-
Srinivasan et al. (2021)	-	-	-	64%$^{(S)}$	-	-	-	-	-
Rubin & Berant (2021)	71%$^{(S)}$	-	-	-	-	-	-	-	-
Scholak et al. (2021)	79%$^{(S)}$	98%	-	-	-	-	-	-	-
GPT-3 13B	16%	43%	0.42	14%	55%	0.51	38%	76%	0.43
" + CSD	20%	66%	0.44	17%	100%	0.48	40%	95%	0.40
" + TST	14%	48%	0.42	-	-	-	60%	88%	0.22
" + CSD + TST	19%	72%	0.43	-	-	-	63%	98%	0.17
GPT-3 175B	28%	49%	0.36	20%	67%	0.36	44%	77%	0.41
" + CSD	35%	73%	0.36	25%	100%	0.32	45%	97%	0.37
" + TST	31%	56%	0.35	-	-	-	60%	88%	0.24
" + CSD + TST	37%	76%	0.34	-	-	-	66%	97%	0.18
Codex 175B	56%	73%	0.25	39%	87%	0.24	45%	79%	0.37
" + CSD	61%	85%	0.23	40%	99%	0.23	46%	97%	0.33
" + TST	60%	81%	0.23	-	-	-	63%	90%	0.21
" + CSD + TST	64%	85%	0.23	-	-	-	63%	99%	0.19

Table 2: Results of each language model on all domains with and without CSD and TST. For SQL, we run the resulting query and report Execution Match accuracy (**Exec.**). For Vega-Lite and SMCalFlow, we instead report Exact Match accuracy (**Acc.**). Edit Distance (**Dist.**) measures average relative edit distance between the prediction and the ground truth. We also report the fraction of **Valid** model outputs (those that parse, type-check and execute). For context only, we show recent results from *supervised* models (trained on the datasets we use) marked with $^{(S)}$.

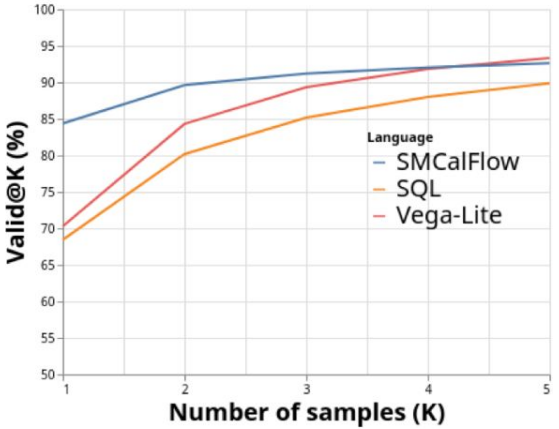
Results



(a)



(b)



(c)

Figure 4: (a) Accuracy and (b) validity of Codex predictions with and without SYNCHROMESH on SMCalFlow as a function of the ground-truth program length. We map program lengths to percentiles, and round to the closest multiple of 10%. Error bands correspond to standard error. (c) Evaluation of the "generate-then-test" approach with Codex, showing the probability of at least one prediction being a valid program (Valid@K) for up to 5 samples.

Results

- SYNCHROMESH brings the output closer to ground truth
- TST and CSD bring complementary benefits.
- SYNCHROMESH adds more value for longer programs
- LLMs augmented with SYNCHROMESH approach but underperform supervised models
- SYNCHROMESH outperforms “generate-then-test”

Список литературы:

- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harri Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021.
- Terence Parr and Kathleen Fisher. Ll (*) the foundation of the antlr parser generator. ACM Sigplan Notices, 46(6):425–436, 2011.
- A Brzozowski. Derivatives of regular expressions. Journal of the ACM (JACM), 11(4): 481–494, 1964.