TempLM: Distilling Language Models into Template-Based Generators

Tianyi Zhang, Mina Lee*, Lisa Li*, Ende Shen*, Tatsunori B. Hashimoto
Computer Science Department, Stanford University
{tz58, minalee, xlisali, endeshen, thashim}@stanford.edu

Abstract

While pretrained language models (PLMs) have greatly improved text generation, they have also been known to produce unfaithful or inappropriate content. In contrast, classic template-based systems provide strong guarantees of faithfulness at the cost of fluency. We propose TempLM, which achieves the best of both worlds by distilling a PLM into a template-based generator. On the E2E and SynthBio data-to-text datasets, we show that TempLM is more faithful than the original PLM and is more fluent than prior template systems. Notably, on an out-of-domain evaluation, TempLM reduces a finetuned BART model's unfaithfulness rate from 83% to 0%. In a human study, we find that TempLM's templates substantially improve upon human-written ones in BERTScore.

1 Introduction

Pretrained language models (PLMs; Brown et al., 2020; Lewis et al., 2020) can generate fluent text and are data-efficient when being transferred to downstream tasks (Chen et al., 2020; Schick and Schütze, 2021). However, PLMs have been known to produce unfaithful outputs (Maynez et al., 2020) and inappropriate content (Gehman et al., 2020) that can lead to disastrous outcomes in real-world deployments (Wired, 2021). These errors can be worsened when models are queried with out-of-domain (OOD) input. Figure 1 shows that querying a finetuned PLM with a novel entity (e.g. Starbucks) not in the training data can lead to surprising failures even though the PLM achieves high in-domain performance. This poses a great challenge in deploying PLMs in real-world applications.

In stark contrast, classic template-based systems (Reiter and Dale, 1997; Barzilay and Lee, 2003; Angeli et al., 2010) employ templates consisting of words and nonterminal fields, which are robust to novel entities by design. Moreover, templates are directly readable by humans, and human inspection can provide direct guarantees of faithfulness. However, templates can be too rigid and produce disfluent text with unexpected inputs. In this work, we seek to borrow the merits of classic template-based techniques to improve faithfulness and interpretability, while retaining the PLM's flexibility and data efficiency.

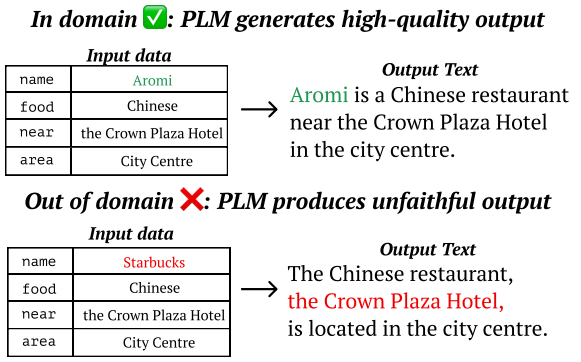


Figure 1: A high-performance PLM finetuned on the E2E dataset generates unfaithful outputs when given out-of-domain inputs. We show later that BART produces such errors 83% of the time while TempLM never suffers from such failures.

We propose TempLM, a novel framework that *distills* a PLM into a template-based system for data-to-text tasks. At training time, TempLM extracts templates that maximally recover the induced probability distribution of the PLM, similar to model distillation (Hinton et al., 2015). At inference time, TempLM uses the PLM to select appropriate data (content selection) and templates (surface realization).

While distilling a PLM into a template-based generator brings benefits, it also raises new challenges. Extracting templates that match a PLM's probability distribution is a challenging combinatorial optimization problem with no clear solution. Our approach relies on two new ideas. First, because our goal is to recover the PLM's induced probability distribution, TempLM initializes its search procedure by *delexicalizing* PLM's genera-

Findings of the Association for Computational Linguistics: ACL 2023, pages 1970–1994
July 9-14, 2023

tion outputs, *i.e.* abstracting the value in the output with data fields. For example, we can delexicalize "Aromi is a Chinese restaurant" into "`[name]` is a `[food]` restaurant." Second, TempLM leverages the PLM's generation ability to refine templates, using a novel *consensus beam search* algorithm. Unlike prior works (Wiseman et al., 2018), our approach can leverage any PLM to generate templates, allowing us to take advantage of improvements in the data efficiency and fluency of PLMs.

We evaluate TempLM on the E2E (Novikova et al., 2017) and the SynthBio datasets (Yuan et al., 2021). We observe that TempLM is the most faithful generation method (with zero faithfulness errors) on the E2E in-domain test set. Furthermore, TempLM fixes the unreliable OOD behavior of PLMs, reducing the unfaithful output rate from 83% to 0%. In addition, we show that TempLM achieves higher metric scores than classic text generation techniques and a previous hybrid neural-template method (5 BLEU scores higher than Wiseman et al. (2018) even when trained with 42 times less data). We further conduct a human study where we ask annotators to write templates for SynthBio with a time constraint. We observe that TempLM produces more fluent templates than both the average template writer and an ensemble aggregating all the template writers.

2 Related Works

PLMs for language generation. PLMs (Radford et al., 2019; Brown et al., 2020; Lewis et al., 2020) are pretrained over large scale text corpora and have significantly improved generation fluency and data efficiency. However, PLMs can still produce unreliable outputs, including hallucination (Maynez et al., 2020), inconsistency (Elazar et al., 2021), toxicity (Gehman et al., 2020), or privacy violations (Carlini et al., 2021). TempLM addresses these shortcomings by distilling a PLM into a less expressive but more trustworthy template-based system, while retaining fluency and data efficiency.

Classic template-based methods. Classic template methods often delexicalize the training set data, *i.e.* they abstract the values in examples from the training data with the nonterminal data fields (Ratnaparkhi, 2002; Oh and Rudnicky, 2000; Rudnicky et al., 1999; Angeli et al., 2010). For example, "The restaurant name is Aromi" can be delexicalized into "The restaurant name is `[name]`." However, delexicalization can be challenging for human-written text. When describing that the customer rating is "3 out of 5," human writers may paraphrase it into "3 stars" or "average." Delexicalization has difficulties capturing this paraphrasing problem and often leaves lexicalized values in templates, which makes the templates less generalizable. In contrast, TempLM first finetunes a PLM on the data-to-text task and then exploits the PLM's ability in smoothing the text distribution to tackle the paraphrasing problem. This technique enables TempLM to generate more fluent outputs than classic template-based systems.

Hybrid neural generation methods. There have been many works that explore different ways to leverage intermediate representations/operations to guide neural generation, including designing an explicit planning module (Puduppully et al., 2019), editing exemplar training examples (Wiseman et al., 2021), and inducing latent variables (Wiseman et al., 2018; Li and Rush, 2020; Ye et al., 2020). Much like classic template-based methods, these systems attempt to learn structured representation from diverse human-written text, which is challenging and often requires heuristics for additional supervision. We differ from prior methods in two important aspects: first, TempLM's templates consist of terminal words and nonterminal fields, which make the templates robust and interpretable. Second, TempLM can leverage any PLM to generate templates, allowing us to take advantage of improved fluency and data efficiency brought by PLMs.

3 TempLM: Template-Based Generators

3.1 Problem Statement

We are interested in data-to-text tasks (Figure 3), where we are given input data d, consisting of *field* and *value* pairs where a field may correspond to multiple values. For example, d could be `{name: [Aromi, aromi], article: [a, an]}`, where `name` is a data field corresponding to multiple values "Aromi" and "aromi". Note that we differ from common data-to-text setups in allowing multiple data values and augmenting d with different capitalization and function words to accommodate for template systems.

Our task is to describe d by some text x generated by $p(x|d)$. To this end, we want to learn a model $p_\theta(x|d)$ using training examples (x, d). In the PLM approach, p_θ is implemented by finetuning a PLM on (x, d), using standard log loss.

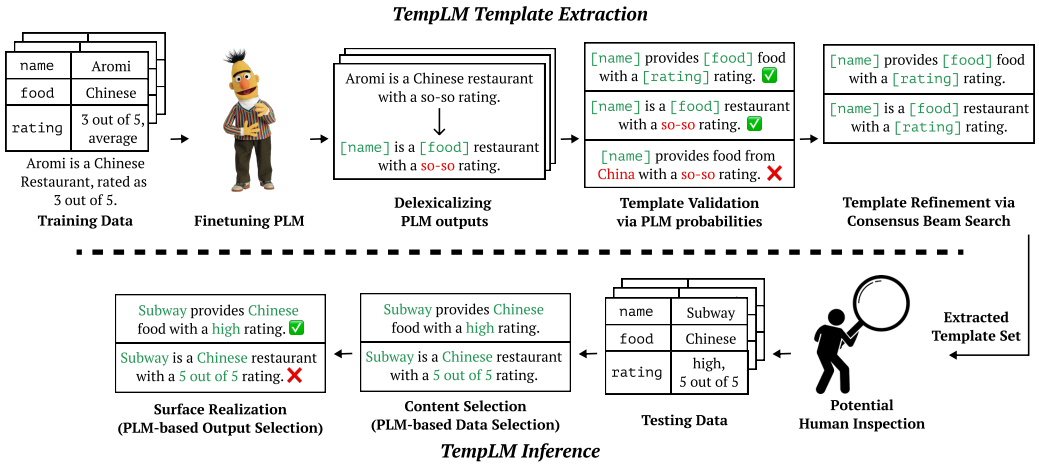


Figure 2: Overview of TempLM. TempLM performs template extraction and inference by treating the finetuned PLM as the ground truth optimization target. We want to extract generalizable templates that contain nonterminal data fields and do not contain lexicalized values.

Helmuth von Schneider

Name	Helmuth von Schneider
Born	04 June 1931, Düren, Germany
Education	German Literature, German Language
Be	{is, are, was, were}
...	...

Bio: *Helmuth von Schneider was a German Editor and writer best known for his novel "Der Heilige Gral" and "Reise der Harlekin". Born on June 4, 1931 in Düren, Germany to parents Anna and Anton Schneider, Schneider attended the University of Düren, where he studied German literature and German Language. [...] He died in a car accident on May 13, 1999 in Düren. Schneider was married to Regina Schneider, and the couple had no children.*

Figure 3: Example of the SynthBio data-to-text task. We are given Wikipedia-style data d about a person and are tasked with generating the biography x.

In template-based generation, we want to obtain a template set T consisting of templates t and ensure that for new input data d, we can generate a high-quality output x. We define a template t as a sequence of *terminal* tokens and *nonterminal* fields that can be replaced by their values in d. For example, a template "The restaurant name is `[name]`" can be filled in as "The restaurant name is Aromi". We represent the action of filling in a template t with data d as $x = F(t, d)$.

A set of templates T captures the data distribution well if at least one template from t is high-quality for every input d. We formalize this goal by stating that for a given input d, we are interested in maximizing $\max_{t \in T} \log p(F(t, d)|d)$. Because we want templates to be inspectable by humans, we want to limit the size of T by a budget B, $|T| \leq B$. Putting these constraints together, we have the following optimization problem:

$$\underset{T,|T|\leq B}{\operatorname{argmax}} \ \mathbb{E}_d[\max_{t \in T} \ \log p(F(t, d)|d)]. \quad (1)$$

What are the implications of Equation (1)? Equation (1) suggests that we would prefer **generalizable templates** such that a single t can be flexibly filled in so that $\log p(F(t, d)|d)$ is high for many different d. In practice, this means that our objective prefers templates with few or no *lexicalized values*. Compare the two templates, "The restaurant name is Aromi" versus "The restaurant name is `[name]`". Equation (1) would prefer the latter template because the first one does not work well when d describes a different restaurant name.

Although Equation (1) nicely captures our intuition of a generalizable template, it presents several optimization challenges. Equation (1) is a size-constrained combinatorial problem that does not have a clear solution. Analyzing the structure of Equation (1), we can decompose it into two separate maximization problems. First, we have the **template extraction** problem of identifying the best template set $\operatorname{argmax}_{T,|T|\leq B}$. Second, given a template set T, we have the **template inference** problem of identifying the best template $\max_{t \in T}$. In the next two sections, we discuss how to leverage PLMs to solve these two problems respectively.

3.2 Template Extraction

The inherent challenge of template extraction is that human-written text in the form of $x \sim p(x|d)$ may not follow a template structure. This is especially true when humans paraphrase the same data value differently, but it could also occur as human-written texts have complex syntactic structures that are not covered by templates. This linguistic diversity makes delexicalization, and more generally learning templates from x, extremely challenging.

Our objective in Equation (1) addresses this key problem. Maximizing $\log p(F(t,d)|d)$ is equivalent to asking for a template t to match *at least one* high probability sequence under p, rather than matching *all* high probability sequences, as is typical in delexicalization or latent-variable based template models. While this approach resolves the paraphrasing problem, it relies upon the true data-generating probability $p(F(t,d)|d)$ which we cannot evaluate. Therefore, we propose to approximate p with a PLM p_θ. This amounts to treating p_θ as the ground truth optimization target, similar to model distillation (Hinton et al., 2015).

While targeting p_θ makes the optimization problem easier, Equation (1) is still intractable because of its difficult combinatorial structure. We design a series of approximations to circumvent the optimization difficulty (Figure 2).

Clustering. Suppose we can obtain the optimal template set $T^* = \{t_1^*, \ldots, t_i^*, \ldots, t_B^*\}$. Then we can identify a cluster function C^* where $C^*(d) = i$ returns the index of the optimal template t_i^* for example d. With C^*, we can decompose Equation (1) into B subproblems that are easier to solve,

$$\underset{t_i}{\operatorname{argmax}} \underset{d \text{ s.t. } C^*(d)=i}{\mathbb{E}} [\log p_\theta(F(t_i,d)|d)]. \quad (2)$$

While obtaining C^* is impossible, we can design approximate clusters C based on the presence of different fields, as is standard in other data-to-text methods (Wiseman et al., 2021).

Delexicalizing PLM outputs. Equipped with approximate clusters C, how can we find templates that work for all examples in the same cluster? Because we are optimizing for p_θ, one natural starting point is to delexicalize the model beam search output x_θ. We denote $t_\theta^{\texttt{delex}}(d)$ as the template we obtain from delexicalizing the PLM output x_θ of the input d and denote $T_\theta^{\texttt{delex}}(d)$ as the corresponding template set.

Delexicalizing x_θ also allows us to be more data efficient and robust. This is because obtaining $T_\theta^{\texttt{delex}}(d)$ only requires unlabeled inputs d as opposed to requiring full supervision (x,d). Obtaining unlabeled data for out-of-domain inputs is substantially easier, and this allows us to exploit data beyond the training set. In practice, we perform data recombination (Jia and Liang, 2016) to not only increase the quantity of d but also explore more field and value compositions.

Template validation via PLM probabilities. While $T_\theta^{\texttt{delex}}(d)$ provides a good initial template set, some of these templates may contain a substantial number of lexicalized data values. To remove these less generalizable templates and fulfill the template budget constraint B, we want to filter the template set $T_\theta^{\texttt{delex}}(d)$. We leverage the PLM's probability estimates to evaluate the template *generalizability*, defined as a template's average log probability over the entire cluster. For a template generated by delexicalizing d, this objective can be written as

$$\sum_{d' \text{ s.t. } C(d')=C(d)} \log p_\theta(F(t_\theta^{\texttt{delex}}(d), d')|d'). \quad (3)$$

where d' are examples sampled from the same data cluster, $C(d') = C(d)$. Equation (3) assigns a scalar value to each $t_\theta^{\texttt{delex}}(d)$ that we use to filter out any ungeneralizable templates. In practice, we retain the top-K best templates in each cluster to form the template set.

Template Refinement via Consensus Beam Search. If a template contains only a few lexicalized values, we can further identify these spans

Algorithm 1 **Consensus Beam Search**

k: beam size, M: maximum length
$\mathcal{V}$: terminal tokens, $\mathcal{V}_T$: nonterminal fields
N: number of inputs
t': partial template where ungeneralizable spans are removed
x_i': $F(t', d_i)$, d_i: ith input data
d_i.get($\cdot$): return the best value token for a field token

1: $B_0 \leftarrow \{\langle 0, \text{BOS}\rangle\}$
2: **for** $t \in \{1, \ldots, M-1\}$ **do**
3: $\quad H \leftarrow \varnothing$
4: $\quad$ **for** $\langle s, \mathbf{y}\rangle \in B_{t-1}$ **do** # Expansion.
5: $\quad\quad$ **for** $y \in \mathcal{V} \circ \mathcal{V}_T$ **do**
6: $\quad\quad\quad S \leftarrow \varnothing$
7: $\quad\quad\quad$ **for** $i \in \{1, \ldots, N-1\}$ **do**
8: $\quad\quad\quad\quad$ **if** $y \in \mathcal{V}$ **then**
9: $\quad\quad\quad\quad\quad S$.add($\log p_\theta(\mathbf{y} \circ y | x_i', d_i)$)
10: $\quad\quad\quad\quad$ **else** # Field token substitution.
11: $\quad\quad\quad\quad\quad S$.add($\log p_\theta(\mathbf{y} \circ d_i$.get$(y) | x_i', d_i)$)
12: $\quad\quad\quad\quad$ **end if**
13: $\quad\quad\quad$ **end for**
14: $\quad\quad\quad s \leftarrow S$.avg() # Aggregation.
15: $\quad\quad\quad H$.add($\langle s, \mathbf{y} \circ y\rangle$)
16: $\quad\quad$ **end for**
17: $\quad$ **end for**
18: $\quad B_t \leftarrow H$.topk(k)
19: **end for**
20: **return** B_t.max()

Algorithm 1 : We search for a common constituent $\mathbf{y}$ that can be infilled to all partial descriptions x_i'. In contrast to conventional beam search, we aggregate the log probability scores across different inputs at each step (Line 6 to Line 14). To generate nonterminal fields (*e.g.* `[name]`), we account for how they will be filled in with different input d_i' in Line 11.

using a token-level version of Equation (3) and then replace ungeneralizable spans by executing a search algorithm with Equation (3) as the objective. To identify the ungeneralizable spans, we begin by evaluating the token-level equivalent to Equation (3) (see Appendix A.1 for details). We then aggregate these token-level scores into a constituent-level score using a constituency parser, and mark any constituent whose score is lower than a threshold as ungeneralizable.

To salvage these ungeneralizable spans, we leverage a PLM to optimize for Equation (3) directly. We remove the ungeneralizable spans to form partial template x' and learn an infilling model $p_\theta^{\text{infill}}(x|x', d)$ to replace the ungeneralizable spans. We implement p_θ^{infill} by finetuning a different PLM and present the details in Appendix B.3.

There are two challenges we face in optimizing Equation (3). First, the infilling model p_θ^{infill} is learned to generate text, not templates. Second, Equation (3) is an unusual objective in text generation that is a mixture-of-experts of many language models where each model conditions on some input d'. We propose two modifications to the standard beam search algorithm to address these challenges (Algorithm 1). First, we empower the infilling model p_θ^{infill} with the ability to generate nonterminal data fields and define their scores based on how they will be filled in (Line 11). Second, we search for a common output that is the "consensus" of many inputs d' by aggregating the log probability scores across inputs at each decoding step (Line 6 to Line 14). Empirically, we find that template refinement can correct for errors in the earlier steps by removing lexicalized values or incorrect fields in the template. We present a qualitative study of template refinement in Appendix B.3.

Human Inspection and Validation. Once templates are refined, we save them as an internal part of TempLM and use them for template inference at test time. To obtain an even stronger faithfulness guarantee, we can have human inspectors validate each template. TempLM offers two advantages for such human-in-the-loop inspection. First, templates in TempLM are readable by humans. Second, TempLM by design has limited freedom during inference: an output can only be generated from filling in a template with input data. As long as none of the templates contains hallucination or inconsistency, TempLM will be guaranteed to return a faithful output. The combination of interpretability and restricted output space enables a natural interface for human-in-the-loop cooperation, where a human inspector can sanitize all the templates before deploying TempLM into real-world applications.

3.3 TempLM Template Inference

Given the template set T that we extracted, we now need to solve the problem of identifying the best template $\max_{t \in T}$ for a new input d. In TempLM, we leverage PLMs as a core primitive in both the content selection and surface realization steps.

Content Selection requires us to substitute a nonterminal field with the most appropriate value among the multiple values that a field corresponds to. We perform this step using a left-to-right autoregressive PLM. At each decoding step, we directly copy from t when encountering a terminal word; otherwise, we select the most probable data value to replace a field. PLMs are typically trained with byte-pair encoding (Sennrich et al., 2016), which might break up data values into multiple tokens. Performing an exact search involves computing the probability of each multi-token value by additional roll-outs, which slows down inference. We circumvent this problem by performing a greedy search on the first token, which leads to faster or on-par inference time with standard PLM inference.

Surface Realization requires us to select the most appropriate output after templates are filled in. We perform this step by computing $F(t, d)$ for all templates in the same cluster $C(d)$ and returning the one with the highest $p_\theta(F(t, d)|d)$.

4 Experiments

We evaluate TempLM's ability to generate faithful and fluent text in three settings: an in-domain evaluation on standard data-to-text benchmarks, an out-of-domain evaluation that stress tests the ability to generalize to novel inputs, and a human study comparing TempLM's template extraction ability to that of human template writers.

4.1 Experiment Setup

Datasets. We consider two data-to-text datasets: E2E (Novikova et al., 2017) and SynthBio (Yuan et al., 2021). The E2E dataset contains data entries about restaurants and asks for text descriptions of restaurant data. Originally, the E2E dataset contained $42K$ training samples with eight distinct fields and 109 field combinations. To better evalu-

ate data efficiency and faithfulness, we downsample the training set to ten samples per field combination. Results on the full E2E dataset are similar and are shown in Appendix B.3. We evaluate on the official validation and test sets.

SynthBio asks systems to write biographies based on Wikipedia-style data tables and was originally proposed as an evaluation set for WikiBio (Lebret et al., 2016). Because WikiBio is a noisy dataset created by automatic retrieval and contains pervasive hallucinations, we decided to use SynthBio instead, by splitting it into training, validation, and test sets, and evaluate on the test set. We summarize the dataset statistics in Table 5.

Evaluation Metrics. We evaluate the fluency of the generated outputs by reference-based evaluation. For E2E, we use the official toolkit and evaluate in terms of BLEU (Papineni et al., 2002), NIST (Belz and Reiter, 2006), ROUGE-L (Lin and Rey, 2004), CIDEr (Vedantam et al., 2015), and METEOR (Banerjee and Lavie, 2005). For SynthBio, we evaluate by BLEU, ROUGE-L, and BERTScore (Zhang et al., 2020).

On the E2E dataset, we also evaluate the faithfulness of a system output. We define an output description to be faithful if it does not contradict the input data or hallucinate information not present in the input. To automatically evaluate this, we manually inspected system output descriptions in the validation set and collected common paraphrases of each possible data value. For example, a customer rating of "3 out of 5", may appear as "3 stars", "average", etc. This allows us to develop a matching-based metric: we count precision error $E_{\text{precision}}$ when a piece of system output contains any paraphrase that matches with a value not in the input (hallucination) or a value different from the one provided in the input (inconsistency).

Note that $E_{\text{precision}}$ is a conservative metric. When we encounter novel phrasings that do not match any entry in our phrasing collection, we do not count them toward $E_{\text{precision}}$. We present more implementation details in Appendix B.2. For template-based methods, we reuse the same routine to measure the percentage of templates that contain lexicalized values (%. Lex. Temp), which measures the generalizability of the templates. We calculate an analogous recall-oriented metric E_{recall} and provide the results in Appendix B.3. We focus on $E_{\text{precision}}$ instead of E_{recall}, as E2E does not require systems to verbalize every value in d.

Implementing TempLM. We implement $p_\theta(x|d)$ and the infilling model $p_\theta(x|x', d)$ by finetuning $\text{BART}_{\text{BASE}}$ (Lewis et al., 2020). On E2E, we assign training samples that have the same combination of fields into the same cluster, which results in 109 clusters. We use data recombination (Jia and Liang, 2016) to combinatorially create 50 samples for each cluster and thereby increase the training data size by five times for template extraction. We define the target number of templates per cluster for TempLM to be five, which results in around 500 templates after deduplication. On SynthBio, we cluster data by the "occupation" field, which results in eight clusters, and we set the TempLM's budget to be ten templates per cluster. We do not perform any data augmentation for SynthBio. More training details are described in Appendix B.2.

Baselines. We compare to three classes of baselines. To compare to existing PLMs, we evaluate a finetuned $\text{BART}_{\text{BASE}}$ model and a KGPT model (Chen et al., 2020), which improves a LM by knowledge-grounded pretraining.

For classic template systems that delexicalize training samples, we compare to TempClassic, which delexicalizes the training data but uses our PLM based inference procedure. We also compare to the SUB baseline (Wiseman et al., 2018), which replaces the PLMs based inference in TempClassic with a rule-based procedure.

For recent hybrid neural-template methods, we compare to the NTemp method (Wiseman et al., 2018). As we were unable to obtain good performance by NTemp on the downsampled training set, we evaluate the model trained on the full E2E training set.

Finally, we performed ablation studies by removing the template refinement (- Refinement) and template validation (- Validation) components from TempLM.

4.2 In-domain Experiment

Table 1 shows that on E2E and SynthBio, TempLM is more faithful than BART while achieving higher metric scores than other template-based methods.[1]

TempLM is faithful. TempLM is the only method that achieves *zero* $E_{\text{precision}}$ across validation and test sets. This improvement over BART suggests TempLM's usefulness in practice. For real-world deployments, we can further leverage human in-

[1] We present other metric scores and validation set results in Appendix B.3.

	$E_{\text{precision}} \downarrow$	BLEU↑	ROUGE-L↑
BART	6.0 ± 2.9	$\mathbf{66.2} \pm 0.5$	$\mathbf{68.4} \pm 0.7$
TempLM	$\mathbf{0.0} \pm 0.0$	61.5 ± 1.0	64.5 ± 0.8
KGPT	8	58.41	63.93
Neighbor Splicing*	543	24.12	37.46
NTemp†	7	55.17	65.70
TempClassic	46.7 ± 25.4	52.1 ± 2.0	62.2 ± 2.3
SUB	110.7 ± 36.2	45.3 ± 1.9	55.6 ± 2.4

	BLEU↑	BERTScore F1↑
BART	$\mathbf{40.8} \pm 0.2$	$\mathbf{55.2} \pm 0.1$
TempLM	40.3 ± 0.3	54.3 ± 0.1
TempClassic	36.6 ± 0.2	48.8 ± 0.1
SUB	14.1 ± 0.1	18.9 ± 0.1

Table 1: Automatic metrics averaged over three random seeds on the E2E and SynthBio test sets. We bold the best numbers in each column and show standard errors with error bars. First, TempLM produces zero unfaithful outputs on E2E. Second, TempLM achieves better or on-par performance on reference-based evaluation than other template systems. *:We find the specialized training procedure of Wiseman et al. (2021) cannot do well on the subsampled E2E training set. †:We compare to a model trained on the *full* E2E training set, which was released by Wiseman et al. (2018). We were unable to train NTemp models on the subsampled E2E dataset to convergence.

	E2E				SynthBio	
	$E_{\text{precision}} \downarrow$	%. Lex. Temp ↓	BLEU↑	#. Temp ↓	BLEU↑	#. Temp ↓
TempLM	$\mathbf{0.0} \pm 0.0$	$\mathbf{5.2} \pm 1.2$	61.5 ± 1.0	$\mathbf{471.7} \pm 62.9$	$\mathbf{40.3} \pm 0.3$	**80**
- Refinement	$\mathbf{0.0} \pm 0.0$	12.1 ± 1.3	61.4 ± 0.9	534.3 ± 8.5	35.2 ± 0.9	**80**
- Validation	2.7 ± 2.2	21.4 ± 2.6	$\mathbf{64.0} \pm 1.0$	2047.3 ± 43.7	36.4 ± 0.1	1511
TempClassic	46.7 ± 25.4	37.4 ± 0.5	52.1 ± 2.0	978.3 ± 1.2	36.6 ± 0.2	1511

Table 2: Ablation results averaged over three random seeds on different template-based systems. We bold the best numbers in each column and show standard errors with error bars. TempLM extracts most generalizable templates and achieves good performance with a small number of templates.

spection to sanitize TempLM's template set, which allows us to remove any lexicalized values in the templates and obtain a strict guarantee for TempLM's faithfulness. In contrast, TempClassic produces almost eight times more precision errors than BART (46 vs. 6), which shows the difficulty of inducing templates over human-written text.

TempLM is fluent and data-efficient. We observe that on E2E, TempLM achieves higher metric scores than other baselines except BART, and on SynthBio, TempLM even performs similarly to BART despite using the less expressive template representation. This demonstrates that TempLM achieves better fluency than previous template methods and is competitive with neural methods. In addition, TempLM retains the data efficiency of PLMs. In particular, TempLM achieves a significant 5 BLEU score improvement over NTemp, which is trained with much more data (1090 vs. $42K$ training samples). In contrast, the state-of-the-art method Neighbor Splicing cannot do well when trained with only 1090 data points.

TempLM enables trade-offs between fluency, robustness, and interpretability. We designed TempLM to have a small number of templates to make TempLM more conducive to human inspection. TempLM successfully achieves this, using less than 500 templates for E2E and only 80 templates for SynthBio. Comparing TempLM without Refinement and TempLM without Validation, we find that template validation reduces the number of templates and substantially increases reliability (halving the percentage of templates containing lexicalized values), but may incur a minor performance drop in fluency.

We find that the template structure is simpler on E2E, and refinement does not add substantial benefit. However, on Synthbio refinement is critical to reversing the performance drop and results in a 4 BLEU score gain. Upon inspection, we find that template refinement can accurately remove ungeneralizable spans in the longer and more complicated templates, which is necessary for SynthBio.

Overall, we find that TempLM ensures faithfulness, retains the PLM's fluency and data efficiency, and balances between performance and interpretability. In the following sections, we go beyond automatic in-domain evaluation. We first stress test systems with out-of-domain inputs and perform a human study to showcase the difficulty of template extraction.

4.3 Out-of-domain Experiment

Models deployed in real-world applications need to be robust to test distributions different from the training distribution. To test for out-of-domain

	Unfaithful Output Rate (%)
BART	83.3
KGPT	16.6
Neighbor Splicing	100
TempLM	0

Table 3: Human annotated unfaithful output rates in out-of-domain (OOD) evaluation. We observe outputs from other systems exhibit pervasive unfaithful errors whereas TempLM continues to remain faithful.

		BERTScore F1
Writer Cluster	Human	51.3 ± 2.3
	Human Ensemble	54.0
	BART	58.5 ± 0.2
	TempLM	58.8 ± 1.0
Spy Cluster	Human	42.2 ± 4.4
	Human Ensemble	48.5
	BART	55.3 ± 0.1
	TempLM	50.8 ± 0.9

Table 4: Human study results on two clusters of the SynthBio test set. Human-written templates result in low metric scores even in the ensemble setting, showcasing the difficulty of identifying distributional characteristics for human and the efficacy of TempLM.

(OOD) generalization, we simulate such a setting on E2E by testing models with entities that are not seen during training.

We create our OOD evaluation by taking fields in E2E (`area`, `eatType`, `food`, `name`, `near`) and filling in common entities scraped from the internet to create 54 novel examples. For instance, we create examples like {`area: Central Park, name: McDonald's, ...`}. We inspect the system outputs manually to check the correctness and present the results in Table 3. We observe that outputs from other systems produce are frequently unfaithful, often confusing entities from different types. In the previous example, BART mistakenly outputs "Central park is a restaurant ...", confusing `area` with `name`. In contrast, TempLM is robust to novel inputs and does not produce *any* unfaithful outputs. We provide the list of novel entities used in creating OOD input and more qualitative examples in Appendix B.4.

4.4 Human Study

To demonstrate the difficulty of generating templates, we conduct a human study on two clusters of the SynthBio dataset. We recruited ten volunteers from our institution to be our template writers and assigned five writers to work on each cluster. Each template writer was given thirty minutes to write templates, and they wrote eleven templates on average. We presented them the same data that TempLM operated on: roughly 200 training examples per cluster, including the input data d and associated text x. We include our human study instruction and interface in Appendix B.5.

To evaluate human performance, we used the human-written templates in our LM-based inference pipeline and measured automatic metric scores. Table 4 shows the BERTScore F1 for both the average template writer as well as an ensemble of five template writers. We report other metric scores in Appendix B.5. We observe that the templates extracted by TempLM lead to better performance than the human-written ones, indicating the intrinsic difficulty of template writing. Based on observing template writers during the writing process, we found that a common strategy is to first go through a subset of the training examples and then find canonical examples to delexicalize. However, we identified a few shortcomings. First, our writers typically only read a few examples (approximately 5 to 20) before they exhaust their cognitive load. As a result, some writers fail to write templates that capture the less common examples. Second, our volunteers may fail to pick the more canonical examples and choose to delexicalize examples that are not the most generalizable. Although well-trained template writers with domain knowledge might have written better templates, the difficulty in identifying such distributional characteristics remains true for any sizable data.

5 Conclusion and Future Work

We propose TempLM, a novel framework for distilling PLMs into template-based systems. TempLM is designed to achieve better robustness and interpretability while inheriting the fluency and data efficiency of PLMs. Our evaluations show that TempLM can completely eliminate the unfaithful outputs produced by a finetuned BART model for out-of-domain inputs. On in-domain evaluation, TempLM is able to produce more fluent outputs compared to classic template systems, prior neural-hybrid template methods, and even human template writers. In the future, we look forward to extending the TempLM framework to learn compositional templates and grammars, as well as improving its coverage to diverse outputs, potentially via paraphrases of its input data.

Limitations

Our system distills PLMs into a less expressive but trustworthy set of templates. In developing this method, we explicitly trade off linguistic diversity for faithfulness guarantees. While this approach works well on academic benchmarks, in more complicated real world settings sacrificing linguistic diversity may impact different groups to a different extent. This raises the question of fairness and we hope to investigate such problems on more realistic datasets in future work.

References

Gabor Angeli, Percy Liang, and Dan Klein. 2010. A simple domain-independent probabilistic approach to generation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 502–512, Cambridge, MA. Association for Computational Linguistics.

S. Banerjee and A. Lavie. 2005. METEOR: An automatic metric for mt evaluation with improved correlation with human judgments. In *Association for Computational Linguistics (ACL)*.

Regina Barzilay and Lillian Lee. 2003. Learning to paraphrase: An unsupervised approach using multiple-sequence alignment. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 16–23.

Anja Belz and Ehud Reiter. 2006. Comparing automatic and human evaluation of NLG systems. In *11th Conference of the European Chapter of the Association for Computational Linguistics*.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. 2021. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650.

Wenhu Chen, Yu Su, Xifeng Yan, and William Yang Wang. 2020. KGPT: Knowledge-grounded pre-training for data-to-text generation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8635–8648. Association for Computational Linguistics (ACL).

Yanai Elazar, Nora Kassner, Shauli Ravfogel, Abhilasha Ravichander, Eduard Hovy, Hinrich Schütze, and Yoav Goldberg. 2021. Measuring and improving consistency in pretrained language models. *Transactions of the Association for Computational Linguistics*, 9:1012–1031.

Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. 2020. RealToxicityPrompts: Evaluating neural toxic degeneration in language models. pages 3356–3369.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Association for Computational Linguistics (ACL)*, pages 12–22. Association for Computational Linguistics.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Association for Computational Linguistics (ACL)*, pages 2676–2686.

Rémi Lebret, David Grangier, and Michael Auli. 2016. Neural text generation from structured data with application to the biography domain. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1203–1213.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Association for Computational Linguistics (ACL)*.

Xiang Lisa Li and Alexander Rush. 2020. Posterior control of blackbox generation. In *Association for Computational Linguistics (ACL)*, pages 2731–2743.

C. Lin and M. Rey. 2004. Looking for a few good metrics: ROUGE and its evaluation. In *NTCIR Workshop*.

Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. 2020. On faithfulness and factuality in abstractive summarization. In *Association for Computational Linguistics (ACL)*, pages 1906–1919.

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2017. The E2E dataset: New challenges for end-to-end generation. In *Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pages 201–206.

Alice H. Oh and Alexander I. Rudnicky. 2000. Stochastic language generation for spoken dialogue systems. In *ANLP-NAACL 2000 Workshop: Conversational Systems*.

K. Papineni, S. Roukos, T. Ward, and W. Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. In *Association for Computational Linguistics (ACL)*.

Ratish Puduppully, Li Dong, and Mirella Lapata. 2019. Data-to-text generation with content selection and planning. *AAAI Conference on Artificial Intelligence*.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).

A. Ratnaparkhi. 2002. Trainable approaches to surface natural language generation and their application to conversational dialog systems. *Computer Speech & Language.*, 16:435–455.

Ehud Reiter and Robert Dale. 1997. Building applied natural language generation systems. *Natural Language Engineering*, page 57–87.

Alexander I. Rudnicky, Eric H. Thayer, Paul C. Constantinides, Chris Tchou, Rande Shern, Kevin A. Lenzo, W. Xu, and Alice H. Oh. 1999. Creating natural dialogs in the carnegie mellon communicator system. In *EUROSPEECH*.

Timo Schick and Hinrich Schütze. 2021. Few-shot text generation with natural language instructions. In *Empirical Methods in Natural Language Processing*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Association for Computational Linguistics (ACL)*, pages 1715–1725.

R. Vedantam, C. L. Zitnick, and D. Parikh. 2015. CIDEr: Consensus-based image description evaluation. In *Computer Vision and Pattern Recognition (CVPR)*, pages 4566–4575.

Wired. 2021. It began as an ai-fueled dungeon game. it got much darker. Https://www.wired.com/story/ai-fueled-dungeon-game-got-much-darker/.

Sam Wiseman, Arturs Backurs, and Karl Stratos. 2021. Data-to-text generation by splicing together nearest neighbors. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4283–4299.

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2018. Learning neural templates for text generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3174–3187, Brussels, Belgium. Association for Computational Linguistics.

Rong Ye, Wenxian Shi, Hao Zhou, Zhongyu Wei, and Lei Li. 2020. Variational template machine for data-to-text generation. In *International Conference on Learning Representations*.

Ann Yuan, Daphne Ippolito, Vitaly Nikolaev, Chris Callison-Burch, Andy Coenen, and Sebastian Gehrmann. 2021. Synthbio: A case study in faster curation of text datasets. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*.

A Additional Details on Template Refinement

A.1 Token-level Generalizability Measure

Our goal is to identify a set of generalizable templates given a budget B such that a single t can be flexibly filled in so that $\log p_\theta(F(t,d)|d)$ is high for many different examples d. Equation (3) does this exactly: we fill in a single template t with many other examples d from the same cluster and measure the sum of their log probabilities. We want to generalize Equation (3) to a token-level generalizability measure, which tells us which tokens within a template t will receive a high probability after the template is filled in with new data. Our idea is to align tokens in the template with tokens in the output and aggregate the corresponding token probabilities across many different outputs.

Let us use j as the token index and denote x_j as the jth token in an output text x and t_j as the jth token in a template t. We use $x_{:j}$ to represent the prefix up to the jth token in x and analogously defined $t_{:j}$. We leverage an alignment function $A(t,d,j)$, where $F(t,d)_{A(t,d,j)}$ gives the token that corresponds to t_j after t is filled in. The alignment A handles the discrepancy in length that is caused by the template fill-in process because the fill-in function F substitutes nonterminal fields with various length data given in d. With the help of A, we can define the token-level generalizability for a token t_j as,

$$\sum_{d' \text{ s.t. } C(d')=C(d)} [\log p_\theta(F(t_\theta^{\texttt{delex}}(d)_{A(t,d,j)}, d') | F(t^{\texttt{delex}}, d')_\theta(d)_{:A(t,d,j)})]. \tag{4}$$

Equation (4) provides a token-level measure, which we can easily turn into a span-level measure by calculating the joint token-level probability. We use this idea to calculate the generalizability of nonterminal fields that correspond to values of multiple tokens. Equation (4) gives us an useful tool for telling which tokens are ungeneralizable and we can then leverage the generation ability to replace these tokens by directly optimizing Equation (4).

Now that we formalize token-level generalizability with Equation (4), our plan is to iteratively remove ungeneralizable spans and use an infilling model to generate new template spans. We can decompose this procedure into two subproblems: removing ungeneralizable spans and generating new template spans. We discuss them in the next two sections, respectively.

A.2 Removing Ungeneralizable Spans

The key problem we want to solve in span removal is to group multiple ungeneralizable tokens together and remove them at the same time. This is because if we remove ungeneralizable tokens one at a time, we would still condition on other ungeneralizable tokens, which deteriorates performance in practice. We leverage constituency parsing (Kitaev and Klein, 2018) to solve this problem. For each constituent in the parse tree, we calculate Equation (4) for each token in the constituent and compute the average. We set a threshold and remove all constituents whose generalizability measure is worse than this threshold.

A.3 Generating Template with Consensus Beam Search

We refer to Section 3.2 for the description of our template generation process. In Algorithm 1, we rely on the subroutine $d_i.\text{get}(\cdot)$, which gives us the best data value among the multiple options in d for a nonterminal field. Implementing this subroutine exactly requires us to evaluate all data values at each decoding step, which is computationally expensive. In practice, we perform a greedy selection based on the first token in each data value.

B Additional Details on Experiments

B.1 Dataset Details

We include the dataset statistics of SynthBio and subsampled E2E datasets in Table 5.

B.2 Model Training Details

Left-to-right Autoregressive LM. We finetune a $\text{BART}_{\text{BASE}}$ model to implement $p_\theta(x|d)$. On the downsampled E2E dataset, we train for 10 epochs for a batch size of 16 and a learning rate of 3×10^{-5}.

	# Train	Average Length	# Fields
E2E	1090	19.8	8
SynthBio	2896	93.1	78

Table 5: Statistics of SynthBio and the downsampled E2E dataset.

Data Field	Data Value
article	a, an
be	is, are, was, were
number	one, two, three, four, five, six, seven, eight, nine, ten
pronoun_a	he, she, they
pronounce_b	him, her, them
pronounce_c	his, her, their
relation	son, daughter

Table 6: Data fields and values we used for augmenting SynthBio input.

We train with half precision using the huggingface implementation. On SynthBio, we train for 5 epochs for a batch size of 8 and a learning rate of 3×10^{-5}. We train with half precision using the huggingface implementation.

Infilling LM. We train our infilling models by masking a random 0 to 10 word span and predicting the masked out span. We finetune a $\text{BART}_{\text{BASE}}$ model to implement $p_\theta(x|x', d)$. On the downsampled E2E dataset, we train for 50 epochs for a batch size of 16 and a learning rate of 3×10^{-5}. We train with half precision using the huggingface implementation. On SynthBio, we train for 20 epochs for a batch size of 16 and a learning rate of 3×10^{-5}. We train with half precision using the huggingface implementation.

TempLM. On E2E, we cluster based on field combination. In total, we have 109 clusters and in each cluster, we have 10 training samples. We perform data recombination to create 50 examples for each cluster. Our template validation selects the top 5 templates and performs template refinement on these templates. Our template refinement process uses -2 log probability as a threshold for removing ungeneralizable spans.

B.3 In-domain Evaluation

Additional Details for Experiment Setup. On E2E, the `familyFriendly` field is a binary field with values being either "yes" or "no". To accommodate template-based generation, we replace "yes" with "family friendly" and "family-friendly" and replace "no" with "not family friendly" and "not family-friendly". We augment E2E input d with article words `[article: [a, an]]`.

On SynthBio, we augment inputs with values listed in Table 6. For `article`, `be`, and `number`, we include them as multiple value options in the input. For pronouns and relations, we assign the correct value based on the `gender` field in the input. We parse all dates into day, month, and year and create separate fields to support different data formats in the templates.

Implementation of Faithfulness Evaluation. We present the phrasing collection we used for matching output in Table 7 and Table 8. We use this phrasing collection to perform a matching based faithfulness evaluation. We consider a phrase in an output to have a precision error if it matches with a field and value pair that is not present in the input data. We consider an output as having recall error E_{recall} if we cannot identify any phrase in the output that corresponds to some field and value pair in the input data

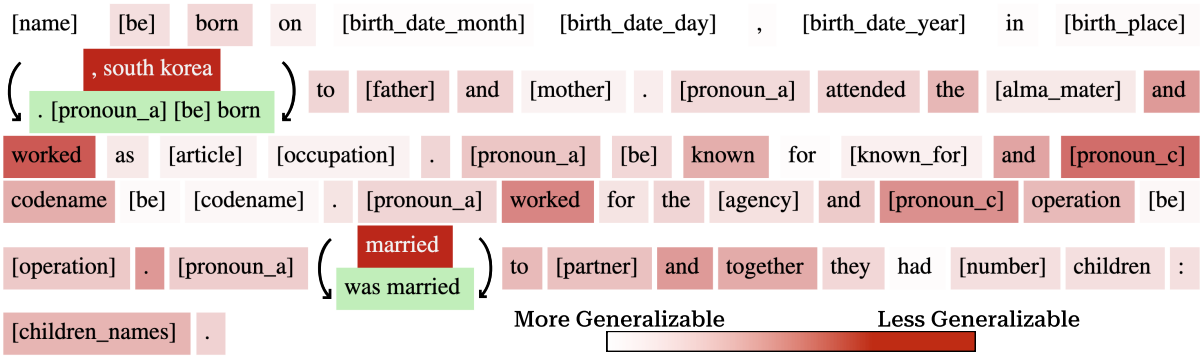


Figure 4: A qualitative example of the TempLM refinement process. We color a terminal word or a nonterminal field as more red if it is less generalizable, measured by the token-level generalizability (Appendix A.1). We mark the refinements TempLM by arrows, coloring the refinement outcome in green.

Because our phrasing collection is imperfect and alternative phrasing may exist, we expect $E_{\text{precision}}$ to be an underestimate and E_{recall} to be an overestimate of actual errors.

Additional Results for Section 4.2. We present a full set of metrics scores for subsampled E2E and SynthBio in Table 9 and Table 10. We make similar observations as in Section 4.2: first, TempLM is the most faithful system on E2E, never producing any precision error; second, TempLM is more fluent than other template systems, achieves better scores with the most of the metrics (BLEU, NIST, CIDEr), and on-par scores with METEOR and ROUGE-L.

We carry out the same experiment on E2E with models trained on the full dataset and present the results in Table 11. We observe that similar to TempLM is the only model that never produces unfaithful on both the test set and the validation set. BART becomes more faithful with more training data. Similar to the experiments on the subsampled training set, TempLM achieves better fluency than NTemp and SUB. One different observation from Table 11 is that TempClassic achieves much better fluency and faithfulness. This is because by leveraging the full training data, TempClassic obtains a large number of templates (39964). While using a large number of templates is helpful, it makes PLM-based inference infeasibly slow, requiring hours of computation to perform inference on the test and validation sets. Having many templates also makes the template set less interpretable by human inspectors. Therefore, we consider TempClassic an impractical baseline.

Qualitative Examples of Template Refinement. To better explain the inner workings of TempLM, we visualize one example of refinement in Figure 4. We color each word according to its generalizability, measured by a token-level generalizability (see Appendix A.1). From Figure 4, we first observe that our generalizability measure is reliable, successfully distinguishing the lexicalized value "south korea" and disfluent span "married" from the rest of the template. Second, we observe that the refinement step correctly fixes both errors by replacing "south korea" with more generalizable, nonterminal fields and inserting "was" to fix the grammatical error. Figure 4 demonstrates the effectiveness of template refinement and helps explain why refinement leads to a substantial performance gain on SynthBio in Table 2.

From Figure 4, we also observe that the words after "and" often appear less generalizable. This is because there are many alternative "branches" that could continue the prefix in these positions and each alternative option will receive a lower probability under a left-to-right PLM $p_\theta(x|d)$. We find that the infilling PLM $p_\theta(x|x', d)$ is robust to these false positives and typically will leave these spans unchanged. This illustrates the benefits of combining a left-to-right and an infilling PLMs in template refinement.

B.4 Out-of-domain Evaluation

Table 12 displays the list of entities we used for creating the 54 OOD examples we used in our evaluation. Table 13 shows example outputs from the BART model finetuned on the downsampled E2E data with

OOD input. We find that BART often confuses the entity in the `area` field with `name` or ignores the input value and hallucinates "city centre."

B.5 Human Study

We present a full list of metric scores that we used to evaluate our human study in Table 14. We have similar observations as in Section 4.4 that TempLM extracts more fluent templates than our template writers. We append our instructions for template writers and screenshots of our interface to the end of this document.

field	value	phrasing
food	Fast food	Fast food fast food
familyFriendly	yes	is family friendly is kid friendly is children friendly is family-friendly is child friendly is a family friendly is a kid friendly is a children friendly is a family-friendly is a child friendly for a family friendly for a kid friendly for a children friendly for a family-friendly for a child friendly
familyFriendly	no	not family friendly not kid friendly not children friendly not family-friendly not child friendly non family-friendly non-family-friendly non family friendly non-family friendly non children friendly non child friendly
customer rating	1 out of 5	1 out of 5 low customer rating one star 1 star
customer rating	3 out of 5	3 out of 5 customer rating is average average customer rating three star moderate customer rating 3 star
customer rating	5 out of 5	5 out of 5 high customer rating five star 5 star

Table 7: A collection of common paraphrases of given input data. We use this phrasing collection to perform a matching-based faithfulness evaluation for E2E. The second half of this table is in Table 8.

field	value	phrasing
customer rating	high	5 out of 5 high customer rating five star 5 star
customer rating	average	3 out of 5 customer rating is average average customer rating three star 3 star
customer rating	low	1 out of 5 low customer rating one star 1 star
priceRange	less than £20	less than £20 cheap low price range low-priced low priced
priceRange	£20-25	£20-25 moderate price range average price range moderately priced moderate prices average priced
priceRange	more than £30	more than £30 high price range high priced expensive price range is high
priceRange	low	low price range low-priced
priceRange	cheap	cheap low price range low priced
priceRange	moderate	moderate price range moderately priced price range is moderate moderate prices average prices
priceRange	high	high price range high priced expensive price range is high

Table 8: A collection of common paraphrases of given input data. We use this phrasing collection to perform a matching-based faithfulness evaluation for E2E. The first half of this table is in Table 7.

Split	Methods	BLEU↑	NIST↑	METEOR↑	ROUGE-L↑	CIDEr↑	$E_{\text{precision}}$ ↓	E_{recall} ↓
Test	BART	$\mathbf{66.2} \pm 0.5$	$\mathbf{8.5} \pm 0.0$	$\mathbf{43.1} \pm 0.2$	$\mathbf{68.4} \pm 0.7$	$\mathbf{2.2} \pm 0.0$	6.0 ± 2.9	$\mathbf{376.3} \pm 48.1$
	TempLM	61.5 ± 1.0	8.0 ± 0.1	41.0 ± 0.8	64.5 ± 0.8	2.1 ± 0.1	0.0 ± 0.0	471.7 ± 62.9
	NTemp†	55.17	7.14	41.91	65.70	1.70	7	539
	TempClassic	52.1 ± 2.0	7.3 ± 0.1	41.7 ± 1.0	62.2 ± 2.3	1.9 ± 0.1	46.7 ± 25.4	451.7 ± 36.9
	SUB	45.3 ± 1.9	6.9 ± 0.2	40.0 ± 0.2	55.6 ± 2.4	1.4 ± 0.1	110.7 ± 36.2	421.0 ± 12.7
Valid.	BART	$\mathbf{70.8} \pm 0.7$	$\mathbf{8.3} \pm 0.1$	$\mathbf{47.0} \pm 0.1$	$\mathbf{72.8} \pm 0.2$	$\mathbf{2.4} \pm 0.0$	5.0 ± 1.5	$\mathbf{182.0} \pm 11.8$
	TempLM	64.8 ± 0.6	8.0 ± 0.0	43.1 ± 0.4	67.8 ± 0.2	2.2 ± 0.0	$\mathbf{0.0} \pm 0.0$	308.7 ± 4.3
	NTemp†	64.53	7.66	42.46	68.60	1.82	7	539
	TempClassic	52.2 ± 0.6	7.2 ± 0.0	40.9 ± 0.2	60.7 ± 0.9	1.7 ± 0.0	92.7 ± 6.1	401.0 ± 13.2
	SUB	43.0 ± 0.4	6.6 ± 0.1	39.4 ± 0.2	55.0 ± 0.4	1.3 ± 0.0	85.3 ± 16.9	409.7 ± 13.7

Table 9: Evaluation of systems trained on the subsampled E2E datasets.

		BLEU	BERTScore F1	ROUGE-L
Test	BART	$\mathbf{40.8} \pm 0.2$	$\mathbf{55.2} \pm 0.1$	$\mathbf{48.4} \pm 0.2$
	TempLM	40.3 ± 0.3	54.3 ± 0.1	48.3 ± 0.1
	TempClassic	36.6 ± 0.2	48.8 ± 0.1	43.1 ± 0.1
	SUB	14.1 ± 0.1	18.9 ± 0.1	26.4 ± 0.1
Valid	BART	$\mathbf{41.7} \pm 0.3$	$\mathbf{55.6} \pm 0.1$	48.8 ± 0.1
	TempLM	41.3 ± 0.2	55.2 ± 0.2	$\mathbf{49.1} \pm 0.2$
	TempClassic	35.1 ± 0.2	47.7 ± 0.1	42.0 ± 0.1
	SUB	14.0 ± 0.1	19.0 ± 0.1	26.4 ± 0.0

Table 10: Automatic evaluation results on the SynthBio test and validation sets.

Split	Methods	BLEU↑	NIST↑	METEOR↑	ROUGE-L↑	CIDEr↑	$E_{\text{precision}}$ ↓	E_{recall} ↓	#. Templates
Test	BART	$\mathbf{67.1} \pm 0.2$	$\mathbf{8.7} \pm 0.0$	$\mathbf{45.2} \pm 0.0$	$\mathbf{69.5} \pm 0.1$	$\mathbf{2.3} \pm 0.0$	$\mathbf{0.0} \pm 0.0$	$\mathbf{110.7} \pm 5.2$	N/A
	TempLM	57.4 ± 0.6	7.6 ± 0.0	41.0 ± 0.3	65.8 ± 0.3	2.0 ± 0.0	$\mathbf{0.0} \pm 0.0$	506.7 ± 15.6	**509**
	NTemp†	55.17	7.14	41.91	65.70	1.70	7	539	N/A
	TempClassic	58.2 ± 0.0	7.5 ± 0.0	43.7 ± 0.0	67.6 ± 0.0	2.2 ± 0.0	$\mathbf{0.0} \pm 0.0$	516.0 ± 1.0	39964
	SUB	36.8 ± 0.2	5.9 ± 0.0	39.5 ± 0.1	51.2 ± 0.2	0.81 ± 1.6	183.7 ± 3.2	416.3 ± 1.5	39964
Valid.	BART	$\mathbf{69.8} \pm 0.1$	$\mathbf{8.4} \pm 0.0$	$\mathbf{47.6} \pm 0.1$	$\mathbf{74.3} \pm 0.1$	$\mathbf{2.5} \pm 0.0$	0.3 ± 0.3	$\mathbf{256.3} \pm 5.8$	N/A
	TempLM	65.5 ± 0.1	7.8 ± 0.0	45.7 ± 0.1	71.9 ± 0.2	2.4 ± 0.0	$\mathbf{0.0} \pm 0.0$	365.7 ± 9.4	**509**
	NTemp†	64.53	7.66	42.46	68.60	1.82	7	539	N/A
	TempClassic	64.6 ± 0.1	7.8 ± 0.0	46.0 ± 0.0	71.3 ± 0.0	2.4 ± 0.0	4.0 ± 0.0	425.7 ± 0.9	39964
	SUB	35.9 ± 0.1	5.6 ± 0.0	38.8 ± 0.1	51.7 ± 0.1	0.73 ± 0.4	136.0 ± 3.8	374.0 ± 1.7	39964

Table 11: Evaluation of systems trained on the full E2E training set.

Area	Times Square, Central Park, Union Station
Food	Thai, Russian, German
Restaurant	McDonald's, Subway, Starbucks, KFC
Near	McDonald's, Subway, Starbucks, KFC

Table 12: List of novel entities used for creating OOD examples.

Input data	{area: Times Square, eatType: restaurant, food: Thai, name: McDonald's, near: Subway}
BART output	Near Subway in the city center is a restaurant that serves Thai food called the Mcdonald's.
Input data	{area: Central Park, eatType: restaurant, food: Thai, name: McDonald's, near: Subway}
BART output	Central park is a restaurant that serves thai food. it is located near subway.
Input data	{area: Times Square, eatType: restaurant, food: Russian, name: McDonald's, near: KFC}
BART output	Near KFC in the city center is a restaurant called the Times Square. it serves Russian food.
Input data	{area: Union Station, eatType: restaurant, food: German, name: Subway, near: Starbucks}
BART output	In the center of the city near Starbucks is a restaurant called Subway.

Table 13: Examples OOD output from the BART model finetuned on the downsampled E2E dataset. We color unfaithful information red.

		BLEU	BERTScore F1	ROUGE-1	ROUGE-2	ROUGE-L
Writer Cluster	Human	37.3 ± 1.5	51.3 ± 2.3	64.5 ± 1.1	41.1 ± 1.6	44.9 ± 1.7
	Human Ensemble	39.1	54.0	63.7	44.1	47.3
	BART	44.0 ± 0.2	58.5 ± 0.2	$\mathbf{70.6} \pm 0.3$	45.8 ± 0.3	50.9 ± 0.2
	TempLM	$\mathbf{44.3} \pm 1.3$	$\mathbf{58.8} \pm 1.0$	68.6 ± 1.1	$\mathbf{46.8} \pm 1.3$	$\mathbf{51.8} \pm 0.7$
Spy Cluster	Human	24.9 ± 2.0	42.2 ± 4.4	54.8 ± 2.0	34.8 ± 0.6	40.5 ± 1.2
	Human Ensemble	32.1	48.5	57.2	37.2	40.7
	BART	$\mathbf{40.5} \pm 0.4$	$\mathbf{55.4} \pm 0.1$	$\mathbf{68.2} \pm 0.4$	$\mathbf{42.7} \pm 0.3$	$\mathbf{46.5} \pm 0.1$
	TempLM	34.4 ± 2.4	50.8 ± 0.9	61.4 ± 0.9	39.8 ± 1.2	44.1 ± 0.4

Table 14: Human study results on two clusters of the SynthBio test set. We observe that human written templates cannot achieve high metric scores even in the ensemble setting, showcasing the difficulty of writing templates and the efficacy of TempLM.

Designing templates for data to text conversion

Goal: **Write (ideally ten or more) templates** that generate realistic biography
Time: 30 minutes

1. What is this task?

Your goal is to write a set of *templates* that can be used to automatically convert data into text. For example, consider this *data* which have three field and value pairs:

Field	Value
name	Ramazan Inal
nationality	Turkish
occupation	writer

In order to automatically generate this *text* from the data:

Ramazan Inal is a Turkish writer.

we can create this template:

[name] is a [nationality] [occupation].

and our system will deterministically replace each field with the value specified in the data.

[name] → Ramazan Inal
[nationality] → Turkish
[occupation] → writer

[name] is a [nationality] [occupation]. → Ramazan Inal is a Turkish writer.

Because we want to make templates *flexible* so that they can account for potential grammatical changes necessary for different values (e.g. “a Turkish writer” vs. “an English writer”), we added these additional fields and possible values to all input data:

Field	Value

be	One of the following: is, are, was, were
article	One of the following: a, an
number	One of the following: One, two, three, four, five, six, seven, eight, nine, ten

Therefore, the final template with these additional fields and values will be:

[name] [be] [article] [nationality] [occupation].

[name] → Ramazan Inal
[be] → is
[article] → a
[nationality] → Turkish
[occupation] → writer

[name] [be] [article] [nationality] [occupation]. → Ramazan Inal is a Turkish writer.

Note that sometimes, *not all fields are used* to generate the text. In the previous example, the number field is not used anywhere in the text, hence no need to be specified in the template.

2. What is the goal?

Given hundreds of pairs of such data and desired texts, your goal is to **write ten or more templates** that *can best represent the given data and text pairs* as well as *can be used to generate realistic biography for new data*.

For example, the previous template can be used with new data to generate biography as follows:

Template:

[name] [be] [article] [nationality] [occupation].

New data:

Field	Value
name	Joseph Duch

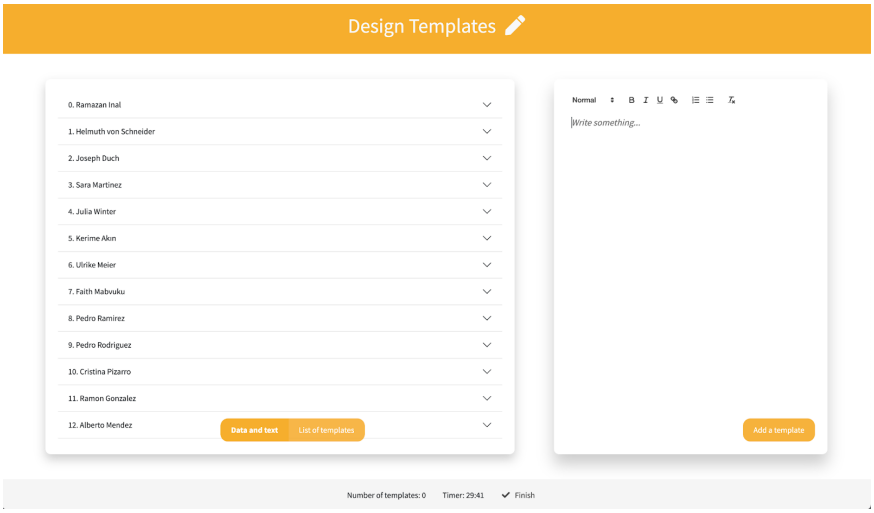
gender	non-binary
nationality	Andorran
occupation	writer
be	One of the following: is, are, was, were
article	One of the following: a, an
number	One of the following: One, two, three, four, five, six, seven, eight, nine, ten

Automatically generated text:

Joseph Duch is a Andorran writer.

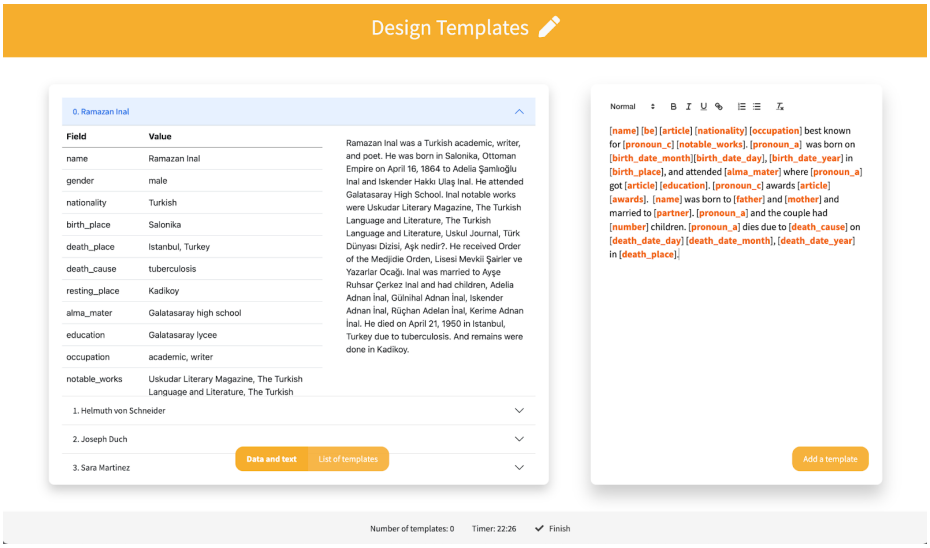
3. How do I do this task?

1. Click one of the links to start: [writer][spy]
    a. Please do not refresh your window! The timer will be reset and you will start over.
    b. We suggest that you maximize the window and zoom out so that you can browse the data easily.

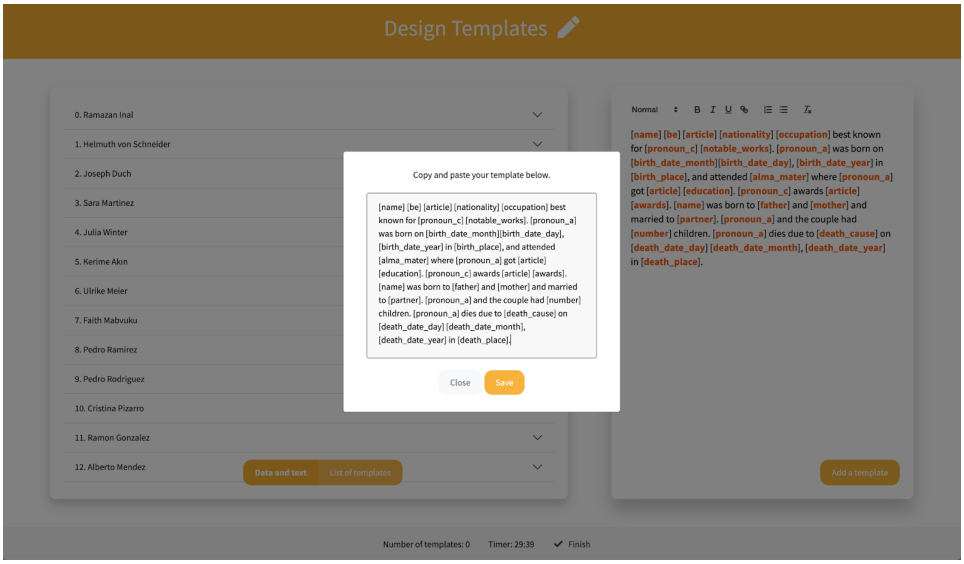


2. In the left panel, you will see all pairs of data and desired texts. Click through them and *get yourself familiar with fields, values, and texts.*
3. In the right panel, you will see a text editor where you can write templates while browsing

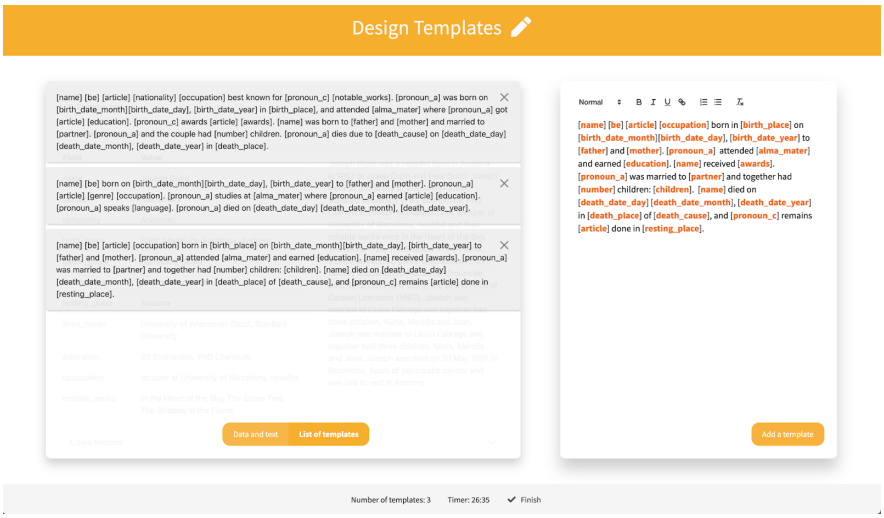
multiple data and desired texts at the same time. Please enclose the field names with brackets (e.g. [name]). Valid field names will be colored in **orange**.



a. Each time you write a template, click the “add a template” button in the right panel, copy and paste your template, and click the “save” button.



b. You can view the list of templates you have written by clicking the “list of templates” button in the left panel.



   c. If necessary, you can delete templates by clicking the close button next to each template in the list.
4. On the bottom of the screen, you will see a counter for the number of templates and a timer.
5. When you are done, click the finish button next to the timer to save your templates. Share the verification code you got with Mina and share the templates you wrote with Tianyi.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
after conclusion before references

☑ A2. Did you discuss any potential risks of your work?
Left blank.

☒ A3. Do the abstract and introduction summarize the paper's main claims?
Left blank.

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☒ Did you use or create scientific artifacts?

Left blank.

☐ B1. Did you cite the creators of artifacts you used?
No response.

☐ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
No response.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
No response.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
No response.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
No response.

☐ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
No response.

C ☑ Did you run computational experiments?

Left blank.

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
section 4.1 specified the kind of model used.

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
section 4.1 discussed experimental setup and section B.2 provides hyperparameter details

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Table 1 and Table 2 provide error bars

☒ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Not in the paper but clear from code release (will be made avaiable after anonymous. period)

D ☑ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☑ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
instruction appended to page 19 onward

☑ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
section 4.4

☑ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
see instruction appended

☒ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
Left blank.

☑ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
see section 4.4