Compute-Efficient Churn Reduction for Conversational Agents

Sarthak Jauhari* **Christopher Hidey***
Google
{jsarthak, chrishidey}@google.com

Abstract

Model churn occurs when re-training a model yields different predictions despite using the same data and hyper-parameters. Churn reduction is crucial for industry conversational systems where users expect consistent results for the same queries. In this setting, compute resources are often limited due to latency requirements during serving and overall time constraints during re-training. To address this issue, we propose a compute-efficient method that mitigates churn without requiring extra resources for training or inference. Our approach involves a lightweight data pre-processing step that pairs semantic parses based on their "function call signature" and encourages similarity through an additional loss based on Jensen-Shannon Divergence. We validate the effectiveness of our method in three scenarios: academic (+3.93 percent improvement on average in a churn reduction metric), simulated noisy data (+8.09), and industry (+5.28) settings.

1 Introduction

Many industry natural language processing systems rely on deep learning models. However, these models can be brittle, leading to different predictions on the same queries after re-training. This is known as **model churn** (D'Amour et al., 2020; Hidey et al., 2022) and can be caused by non-determinism in training due to data ordering and initialization. For conversational agents like Google Assistant or Amazon Alexa, this problem can erode user trust if the system behaves unexpectedly over time.

Common approaches to address this issue include ensembling (Dietterich, 2000), distillation (Hinton et al., 2015; Anil et al., 2020; Kim and Rush, 2016), or co-distillation (Anil et al., 2020; Hidey et al., 2022). All these techniques involve training or serving one or more models in addition to the primary model, and may be unfeasible in practice due to constraints on available compute resources. While training *additional models* provides a regularization effect that results in better robustness to model churn (Hidey et al., 2022; D'Amour et al., 2020), we show that the same effect can be achieved with regularization using *additional data*, which requires no extra compute resources at training or serving.

Function Call Signature	in:get-info-traffic [sl:destination [in:get-location [sl:point-on-map SPAN]]]]
Query 1	traffic to midway airport
Query 2	is traffic heavy on the way to iowa state university

Table 1: Examples from TOP (Gupta et al., 2018) with the same **function call signature**. Intuitively, the representations of the spans for these slots should be close in vector space and the model should be consistent about predicting the same non-span tokens for both queries.

Our approach relies on the observation that similar training examples should have similar representations and predictions. Bahri and Jiang (2021) demonstrated that using nearest neighbors to obtain soft labels for discriminative tasks can reduce churn. However, in structured prediction tasks like conversational semantic parsing, smoothing labels in this way is challenging.

To address this, we propose pairing examples based on their function call signature, i.e. the non-terminal nodes in a parse tree. Table 1 illustrates two queries with the same function call signature. Our churn reduction method consists of two key components: 1) pairing similar examples in a batch based on their function call signature, and 2) introducing a span similarity loss between the slots within a pair. In Table 1, both "midway airport" and "iowa state university" represent locations and should have similar contextual representations in a pre-trained model. Our span similarity loss ac-

*equal contribution

Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track, pages 284–293
December 6-10, 2023

counts for this by using a span encoding to represent these slots and applying a regularizer using a similarity score derived from the Jensen-Shannon Divergence (JSD) of their contextual distributions in the training data.

Our contributions are as follows:

1. We introduce a novel approach to churn reduction by training on examples paired by their function call signature. We regularize slots across the pair using a span similarity loss according to the JSD score.

2. We demonstrate state-of-the-art reduction in model churn on the TOP (Gupta et al., 2018), TOPv2 (Chen et al., 2020), and MASSIVE (FitzGerald et al., 2022) datasets in an "academic" setting with high quality data (+3.93 percent improvement on average in EM@10 as defined in Section 5.1).

3. As industry training sets may be noisy (e.g. due to labeling by a larger model), we also show that these results hold on the "simulated noisy" datasets (+8.09) created by Hidey et al. (2022) and in an industry setting on an internal proprietary dataset (+5.28).

2 Related Work

Model Churn occurs when multiple re-training runs with the same model architecture yield different predictions, even when trained on the same data (Milani Fard et al., 2016). This problem has traditionally been studied for classification tasks (Shamir and Coviello, 2020; Shamir et al., 2020; Jiang et al., 2022; Datta et al., 2023) although we recently explored churn reduction for structured prediction (Hidey et al., 2022).

Techniques such as **ensembling and distillation** are known to improve model calibration (Hansen and Salamon, 1990; Lakshminarayanan et al., 2017) and reduce model churn (Hidey et al., 2022). However, these techniques may not always be practical due to the computational cost – ensembling requires K trained models for inference. Distillation, the process of training a "student" model based on the predictions of a "teacher" (Hinton et al., 2015), traditionally requires only a single model for inference but an ensemble is often used as the teacher (Reich et al., 2020), meaning that the cost of training is the same. Co-distillation (Anil et al., 2020) was developed as an alternative where K peer models are trained in parallel and while training time may be less than Kx this approach still requires a Kx increase in compute resources.

Conversational Semantic Parsing is the task of converting a user query into an executable form that can be understood by a conversational assistant. In our previous work (Hidey et al., 2022), we studied the problem of model churn for this structured prediction task and reported that co-distillation reduces churn on the TOP (Gupta et al., 2018), TOPv2 (Chen et al., 2020), MTOP (Li et al., 2021), and SNIPS (Coucke et al., 2018) datasets for conversational agents. This work is a direct extension of Hidey et al. (2022) - we show that we can reduce model churn using a single trained model, unlike co-distillation which requires training a peer model in parallel. Our approach is similar to the work of Bahri and Jiang (2021) to smooth labels using nearest neighbors. However, their approach does not directly extend to structured prediction tasks. Instead, we create neighbors according to their "function call signature," where the idea is that a trained model should perform the same on training examples with the same shallow structure.

3 Methods

To enhance robustness against churn, we propose two key contributions. First, we balance training data by pairing examples based on their "function call signature," specifically the non-terminal nodes in the semantic parse (e.g. Table 1). We hypothesize that this will improve stability during training by exposing the model to a greater diversity of function call signatures in each batch. Second, we introduce a pairwise similarity loss that aligns spans within the same slot. We achieve this by comparing the distribution of spans across slots using Jensen-Shannon Divergence (JSD), which provides a continuous score for assessing the positive or negative nature of paired spans. We employ these techniques by extending a pointer-generator network with span encodings.

3.1 Baseline Architecture: Pointer-Generator with Span Encoding

Our baseline architecture follows the work of Rongali et al. (2020) and we use a pointer-generator network to make a structured prediction of the semantic parse. This architecture requires an encoded representation of the user query from which to copy. For this representation we use the span encoding

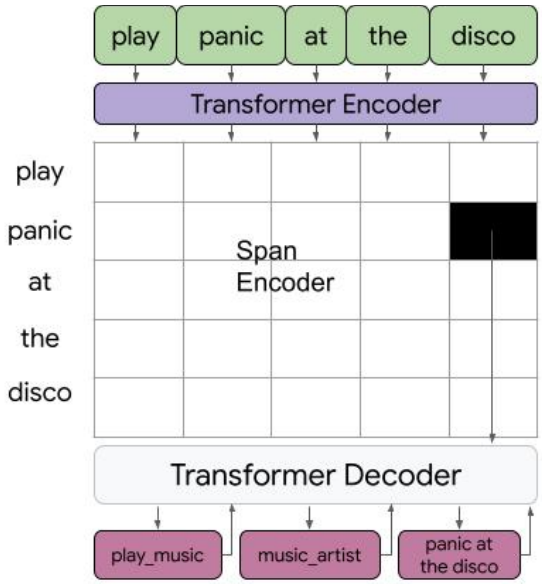

Figure 1: Pointer-generator with span copying.

of Herzig and Berant (2021) (see Figure 1). We theorize that span encodings reduce churn by encouraging spans to be consistently mapped to the same semantic space.

First, we use a transformer encoder to obtain a wordpiece embedding. Then we take the first sub-word embedding to obtain a word-level representation h_i. To compute a span encoding, we concatenate the first and last word embedding in a span and use a dense multi-layer perceptron (MLP) to obtain a unique span encoding:

$$h_{i,j} = MLP([h_i; h_j]) \tag{1}$$

We do this for all $\binom{n}{2}$ spans in the user query.

To obtain the probability of an output token at a given timestep t, we compute the softmax over all possible "copy" spans c_t in the query (the "pointer" in pointer-generator) as well as a fixed vocabulary of intents/slots g_t (the "generator"):

$$p(y_t|\cdots) = \sigma([g_t; c_t]) \tag{2}$$

The vector g_t is computed by multiplying the output vocabulary matrix with the decoder state d_t and passing the result through an MLP. c_t is computed as follows:

$$c_t = [d_t^\top h_{0,0}, d_t^\top h_{0,1}, \ldots d_t^\top h_{0,N}, \ldots d_t^\top h_{N,N}] \tag{3}$$

See Rongali et al. (2020); Herzig and Berant (2021); Held et al. (2023) for further details of this architecture.

During training, we minimize the negative log likelihood of the sequence of copy/generate tokens in the semantic parse:

$$\mathcal{L}_{NLL} = -\sum_{t=1}^{T} \log p(y_t|\cdots) \tag{4}$$

3.2 Pairwise Data

One key observation is that similar queries should have similar semantic parses. For each example in the training set, we compute a "function call signature" by taking all non-terminal nodes in the semantic parse. We then pair examples according to their function call signature as in Table 1.[1] For example, Query 1 and Query 2 have the same function call signature "in:get-info-traffic [sl:destination [in:get-location [sl:point-on-map SPAN]]] " even though in the full semantic parse, SPAN refers to "midway airport" and "iowa state university," respectively.

During training, we group similar examples in batches to encourage stability and balance the types of examples - similar to the well-known approach for classification tasks (Henning et al., 2023). The span encoding representation enables alignment-per-timestep of pairwise examples with the same function call signature[2] (see Table 2), facilitating the approach described in Section 3.3. This approach is similar to that of Bahri and Jiang (2021) but adapted for the task of structured prediction. Rather than using an unsupervised similarity function to obtain soft labels, we use partial labels to obtain similar examples.

However, naively pairing spans based on slots encourages unrelated spans to appear similar, and the default pairing only yields positive examples. For example, in Table 2, "Shinedown" and "Kid Cudi" should have high similarity as musicians, but because "Chicago" is a place, artist, and song it should not be strongly similar to either span due to its multiple meanings. We hypothesize that these issues would cause overfitting between spans and to address them we introduce a "soft" JSD score to better reflect their true similarity.

[1] Some examples will be singletons and are paired with themselves.

[2] Note that with span encodings, two examples with the same function call signature have the exact same target sequence aside from their spans, even with a varying number of tokens per span. Without span encodings, the target sequences would not be aligned.

Query	Target				
can i listen to shinedown ?	[in:play-music	[sl:music-artist-name	shinedown	]	]
i want to hear kid cudi please	[in:play-music	[sl:music-artist-name	kid cudi	]	]
Timestep	0	1	2	3	4

Span	Slots
shinedown	[sl:music-artist-name, [sl:name-event
kid cudi	[sl:music-artist-name, [sl:name-event
chicago	[sl:location, [sl:destination, [sl:source, [sl:location-modifier, [sl:music-artist-name, ...

Table 2: An aligned pair from TOPv2 along with the top occurring spans for the same slot in the training set. Very similar spans will have low entropy in their slot distribution whereas ambiguous spans will have high entropy.

3.3 Span Similarity Loss with JSD

The span distribution is defined as the distribution of slots to which that span has been associated. It was our observation that spans with similar distributions tend to share common characteristics and contextual meaning.

The Kullback–Leibler divergence (KLD) between two discrete probability distributions P and Q is defined as follows:

$$D_{KL}(P\|Q) = \sum_{x\in\mathcal{X}} p(x)\log\frac{p(x)}{q(x)}\,dx \quad (5)$$

The Jensen-Shannon Divergence $D_{JS}(P\|Q)$ is a symmetric version of the KLD metric and is computed by averaging the mixture distributions of $D_{KL}(P\|M)$ and $D_{KL}(Q\|M)$, where $M = \frac{1}{2}(P+Q)$.

In our work, we compute $JSD_{sim} = 1 - D_{JS}(P\|Q)$, where P and Q are the conditional distributions of a slot given a span. For example, let the slot vocabulary be "[sl:location, [sl:destination, [sl:source, [sl:location-modifier, [sl:music-artist-name, [sl:name-event" as in Table 2. Then, given "kid cudi" and "chicago," say that P and Q (computed from counts in the training data) are $[0.01, 0.01, 0.01, 0.01, 0.9, 0.06]$ and $[0.7, 0.1, 0.05, 0.01, 0.13, 0.01]$, respectively. In this case, JSD_{sim} would be 0.58, indicating moderately low similarity.

As spans are potentially many tokens and are thus sparse with very low counts, we compute the overall distribution using **back-off smoothing** (Katz, 1987). We compute the $JSD_{backoff}$ score by averaging the slot distributions for each unigram in a given span. We combine the above two similarities using a hyperparameter, α, which controls the amount of backoff smoothing. The target similarity, y_{sim} is given as follows:

$$y_{sim} = (1-\alpha) * JSD_{sim} + \alpha * JSD_{backoff} \quad (6)$$

In the paired setting, we align the decoder embeddings of spans using the target JSD similarity (y_{sim}). Given aligned spans s_1 and s_2 with span embeddings from Equation 1, we compute a span similarity loss with the mean-squared error between y_{sim} from the JSD score and the predicted $\hat{y}_{sim}$:

$$\hat{y}_{sim} = Linear(h^{s_1}_{i,j}, h^{s_2}_{i,j}) \quad (7)$$

$$\mathcal{L}_{span} = \mathcal{L}_{MSE}(y_{sim}, \hat{y}_{sim}) \quad (8)$$

The overall loss term is then:

$$\mathcal{L} = \mathcal{L}_{NLL} + \lambda\mathcal{L}_{span} \quad (9)$$

4 Experiment Setup

4.1 Datasets

We report results in three settings. First, we conduct our experiments in an **academic** setting using public datasets. To compare directly to our previous work (Hidey et al., 2022), we report results on the TOP (Gupta et al., 2018) and TOPv2 (Chen et al., 2020) datasets. We also run experiments on the more recently released MASSIVE dataset (FitzGerald et al., 2022).[3] These datasets contain hierarchical semantic parses reflecting user intents from domains such as alarms, events, messaging, music, navigation, reminders, timers, and weather. As the training sets were verified by human annotators this setting represents the highest possible data quality. The second setting from our prior

[3]We use only the English partition.

Model	TOP		TOPv2		MASSIVE	
	EM ± STD (@10)	AGR	EM ± STD (@10)	AGR	EM ± STD (@10)	AGR
Baseline (LS)	80.65 ± 0.31 (70.29)	75.48	83.88 ± 0.18 (73.12)	78.15	66.96 ± 0.52 (55.71)	64.49
LS + Pairwise/JSD	**81.27** ± 0.19 **(72.83)**	**78.95**	**84.37** ± 0.10 **(78.21)**	**84.04**	**67.02** ± 0.28 **(56.49)**	**65.37**
CD (Hidey et al., 2022)	81.43 ± 0.41 (73.56)	80.41	84.21 ± 0.10 (76.10)	82.99	**67.00** ± 0.26 **(56.83)**	66.46
CD + Pairwise/JSD	**81.46** ± 0.22 **(74.59)**	**81.87**	**84.63** ± 0.06 **(79.77)**	**86.83**	66.75 ± 0.46 (56.56)	**66.70**

Table 3: Model performance (over N = 10 runs) when trained on **academic** datasets. EM: exact match (mean over 10 runs) with STD (standard deviation). EM@10: EM if all 10 models are correct. AGR: model agreement. **bold:** best setting when controlling for compute resources. Baseline LS and CD results for TOP and TOPv2 are from Hidey et al. (2022).

Model	TOP		TOPv2		MASSIVE	
	EM ± STD(@10)	AGR	EM ± STD(@10)	AGR	EM ± STD(@10)	AGR
Baseline (LS)	78.15 ± 1.23 (61.36)	65.11	81.80 ± 0.25 (67.20)	70.86	**64.25** ± 0.39 (49.26)	56.49
LS + Pairwise/JSD	**80.55** ± 0.15 **(71.53)**	**77.53**	**83.44** ± 0.08 **(76.0)**	**81.6**	63.49 ± 0.23 **(49.83)**	**57.43**
CD (Hidey et al., 2022)	80.83 ± 0.27 (72.14)	78.45	81.97 ± 0.25 (70.12)	75.91	**64.52** ± 0.33 (52.86)	62.1
CD + Pairwise/JSD	**80.86** ± 0.28 **(73.13)**	**80.29**	**83.79** ± 0.07 **(77.79)**	**84.54**	64.34 ± 0.43 **(53.19)**	**62.84**

Table 4: Model performance (over N = 10 runs) when trained on (10%) **systematic noise** datasets. EM: exact match (mean over 10 runs) with STD (standard deviation). EM@10: EM if all 10 models are correct. AGR: model agreement. **bold:** best setting when controlling for compute resources. Baseline LS and CD results for TOP and TOPv2 are from Hidey et al. (2022).

work (Hidey et al., 2022) involves adding **systematic noise** to TOP and TOPv2 with a model trained on 90% of the training data to label the remaining 10%.[4] This setting simulates a "real world" scenario by controlling the amount of noise that enters the training data (e.g. by distilling from a larger language model or another process). Finally the third setting consists of **internal** data, collected from real system output (and therefore noisy) and filtered to match the intents of the aforementioned datasets. The selected domains are a subset of the ones in TOP/MASSIVE - music, alarms, and timers. The data was deduped and the training data was used as-is (noise and all) while the development and test sets were manually re-labeled by human annotators.

4.2 Implementation Details

In order to directly compare to our previous work, we use the same pre-trained 4-layer BERT encoder (Turc et al., 2019; Hidey et al., 2022). For the internal experiments, we use a bespoke 4-layer transformer encoder. The vocabulary was derived from internal data and the model was pre-trained on a large set of system inputs. We report hyperparameters in Appendix B.

[4]We repeat this process for MASSIVE as well.

5 Results

5.1 Evaluation

We evaluate the various methods by re-training each experiment N=10 times on all datasets. As with our previous work, we use the following metrics for evaluation (Hidey et al., 2022): **Exact Match Accuracy (EM)**, the average over N runs of how often the prediction matches the target, **Sequence-Level Model Agreement (AGR)**, the number of times out of N runs that the predictions agree with each other, and **Exact Match Agreement at N (EM@N)**, the number of times out of N runs that the predictions agree *and* match the target. The reason for these metrics is that we want to maximize both accuracy on the gold targets (EM) and consistency across re-training runs (AGR), i.e. minimize model churn. EM@N reflects these dual goals by combining both metrics.

5.2 Comparisons

For our baselines we report results with **label smoothing (LS)** and **co-distillation (CD)**.[5] Label smoothing is often used for calibration of deep learning models (Müller et al., 2020). In this setting, a "soft" target is computed by mixing the true

[5]We used our approach from Hidey et al. (2022) for MASSIVE and the internal dataset but report the numbers verbatim for TOP and TOPv2.

one-hot label with a uniform distribution over all labels. **Co-distillation** was introduced by Anil et al. (2020) and obtained the best results on conversational semantic parsing according to Hidey et al. (2022). Both these approaches are complementary to our methods[6] and for our experiments we combine LS and CD with our approach (**LS + Pairwise/JSD** and **CD + Pairwise/JSD**, respectively). As co-distillation requires additional compute resources, we present both approaches to allow practitioners to make informed decisions given resource constraints.

Model	EM(@10)	AGR
Baseline (LS)	– (–)	–
LS + Pairwise/JSD	+0.17 (+2.39)	+5.28
CD (Hidey et al., 2022)	+0.39 (+0.95)	+2.84
CD + Pairwise/JSD	+0.48 (+1.89)	+4.3

Table 5: Relative results on **internal** dataset with baseline numbers redacted.

5.3 Discussion

Applying the CD/LS + Pairwise/JSD approach shows consistent gains on TOP, TOPv2, and the internal dataset (Tables 3, 4, and 5). There are two noticeable trends. First, **the benefits of our approach become more evident with more data**. TOPv2 has roughly 10 times as many queries as MASSIVE. We obtain the best results relative to the baseline on TOPv2 (+5.09 EM@10) and the internal dataset (+2.39). Intuitively, we are relying on sparse slot distributions which become more reliable in larger datasets at helping the model learn similarity between spans. With enough data LS + Pairwise/JSD can even out-perform the more resource-needy CD setting. Second, the relative gains over the LS and CD baselines are larger when training on the systematic noisy and internal datasets. This suggests that **regularization of spans with a "soft" similarity score encourages robustness to noise**.

We conduct an ablation study on the internal dataset (Table 6) to show the effect of our modeling decisions relative to our LS + Pairwise/JSD model with label smoothing in Table 5 (the low-compute setting with label smoothing only). Removing backoff (setting $\alpha = 0$ in Equation 6) or removing the JSD span loss ($\lambda = 0$ in Equation 9) dramatically degrades the churn metrics. Furthermore, the pairwise setting alone (i.e. simply including similar examples in the same batch) results in a difference of -2.41 and -4.51 EM@10 and AGR, respectively. Finally, we show that the use of span encodings alone is ineffective and is in fact worse than the Baseline (LS) in Table 5.

Model	EM(@10)	AGR
LS + Pairwise/JSD		
-Backoff	-0.3 (-0.01)	-1.14
-JSD	-0.05 (-2.63)	-4.99
-JSD/-Span	-0.14 (-2.41)	-4.51
Span Encoding only	-0.39 (-4.74)	-8.81

Table 6: Ablation Experiments on internal dataset relative to the best low-compute setting.

6 Qualitative Analysis

We conducted an analysis on MASSIVE comparing LS + Pairwise/JSD to LS and noticed better results for calendar/event/date/time intents. [7] Table 7 displays **successful** examples illustrating these observations. For example, given the query "set a reminder about todays faculty meeting at four" on all 10 runs our model (LS + Pairwise/JSD) correctly predicts the target "[in:calendar-set [sl:date todays] [sl:event-name faculty meeting] [sl:time four]]." However, the LS baseline sometimes predicts "[in:calendar-set [sl:date todays] [sl:event-name faculty meeting]]" and misses the date slot. Additionally, it showed improved ability to learn entity identification. Given the query "let me know the recipe for preparing pasta," the baseline may predict "[in:cooking-recipe [sl:ingredient pasta]]" whereas the target is [in:cooking-recipe [sl:food-type pasta]].

We also observed "overtriggering" (i.e. predicting extra slots for argument-less intents), likely due to encouraging span alignment. For instance, the query "is there a chance the hurricane will make landfall" indicates interest in weather conditions rather than a weather event. The correct semantic parse for such queries is [in:unsupported-weather. However, due to the term "hurricane," the model mistakenly predicts [in:get-weather [sl:weather-attribute hurricane]]. [8] Furthermore, the slot "[sl:date-time" was often associated with the intent "[in:update-reminder-date-time" rather than

[6] We use 10% label smoothing for *all* experiments.

[7] Additional results showing the most improved/regressed function call signatures can be found in Appendix D.

[8] See Appendix C for a negative result where we attempted to address this issue.

Query	what time will the soccer match be tonight
Model Run	[in:recommendation_events [sl:event_name soccer match] [sl:timeofday tonight]]
Model Run	[in:calendar_query [sl:event_name soccer match] [sl:timeofday tonight]]
Query	set a reminder about todays faculty meeting at four
Model Run	[in:calendar-set [sl:date todays] [sl:event-name faculty meeting] [sl:time four]]
Model Run	[in:calendar-set [sl:date todays] [sl:event-name faculty meeting]]
Query	show me tomorrows weather in this area
Model Run	[in:weather-query [sl:date tomorrows]]
Model Run	[in:weather-query [sl:date tomorrows] [sl:place-name this area]]
Query	add a reminder of a conference for tomorrow in new york
Model Run	[in:calendar-set [sl:date tomorrow tomorrow] [sl:event-name conference conference] [sl:place-name new york]]
Model Run	[in:calendar-set [sl:date tomorrow tomorrow] [sl:event-name conference conference]]
Model Run	[in:calendar-set [sl:date tomorrow tomorrow] [sl:event-name conference conference] [sl:time]

Table 7: Examples from MASSIVE (FitzGerald et al., 2022) showing model churn in the Baseline (LS) setting but corrected in our model (LS + Pairwise/JSD). The full, correct parse is highlighted in blue. In other cases, model re-training runs result in differences between the prediction and target (inserted/replaced slots highlighted in red. Out of 10 runs, our model always correctly predicts the target.

"[in:update-reminder" (e.g. for "please update my watching the game reminder from 5 pm to 3 pm.")

7 Limitations

As discussed in Section 6, encouraging span alignment may result in the model predicting extraneous slots. On the datasets we used for our experiments, the reduction in churn did not result in a decrease in exact match accuracy. In TOP/TOPv2 and the internal dataset, there are relatively complex function call signatures with many slots and nested intents. However, given a dataset with shallow trees where many examples consist of only a single intent and no slot, there would be no benefit to span alignment and our approach could therefore be detrimental to both churn reduction and accuracy. One possible cause for the limited gains on MASSIVE, other than the dataset size, is that there are no nested intents and relatively simple function call signatures. Thus, practitioners should consider the structure of the semantic parse trees in the training data when considering whether to use this approach.

8 Conclusion: Practical Considerations and Recommendation

When deciding on the best approach for churn reduction, there are three factors to consider: 1) training data size 2) training data quality and 3) compute resource constraints. Given a sufficiently large training dataset, we recommend the LS + Pairwise/JSD approach regardless of the amount of noise in the data. LS + Pairwise/JSD outperformed co-distillation alone on all TOPv2 settings and on the internal dataset. When accounting for resource usage, co-distillation requires 2x resources for training an extra model. Pairwise/JSD requires only an additional preprocessing step over LS/CD and paired data can be cached and updated easily. If there are no resource constraints, CD + Pairwise/JSD can achieve better or comparable results over LS + Pairwise/JSD.

Overall, our approach is especially effective on our internal data due to the large available quantity of un-annotated system output. In a noisy production setting such as ours, we recommend combining our approach with label smoothing or co-distillation, depending on how much data is available and what constraints exist on training time.

9 Acknowledgments

The authors thank the anonymous reviewers for their careful reviews and thoughtful suggestions.

References

Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E. Dahl, and Geoffrey E. Hinton. 2020. Large scale distributed neural network training through online distillation.

Dara Bahri and Heinrich Jiang. 2021. Locally adaptive label smoothing improves predictive churn. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 532–542. PMLR.

Xilun Chen, Asish Ghoshal, Yashar Mehdad, Luke Zettlemoyer, and Sonal Gupta. 2020. Low-resource domain adaptation for compositional task-oriented

semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics.

Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, Maël Primet, and Joseph Dureau. 2018. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces.

Alexander D'Amour, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen, Jonathan Deaton, Jacob Eisenstein, Matthew D. Hoffman, Farhad Hormozdiari, Neil Houlsby, Shaobo Hou, Ghassen Jerfel, Alan Karthikesalingam, Mario Lucic, Yian Ma, Cory McLean, Diana Mincu, Akinori Mitani, Andrea Montanari, Zachary Nado, Vivek Natarajan, Christopher Nielson, Thomas F. Osborne, Rajiv Raman, Kim Ramasamy, Rory Sayres, Jessica Schrouff, Martin Seneviratne, Shannon Sequeira, Harini Suresh, Victor Veitch, Max Vladymyrov, Xuezhi Wang, Kellie Webster, Steve Yadlowsky, Taedong Yun, Xiaohua Zhai, and D. Sculley. 2020. Underspecification presents challenges for credibility in modern machine learning.

Arghya Datta, Subhrangshu Nandi, Jingcheng Xu, Greg Ver Steeg, He Xie, Anoop Kumar, and Aram Galstyan. 2023. Measuring and mitigating local instability in deep neural networks. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 2810–2823, Toronto, Canada. Association for Computational Linguistics.

Thomas G. Dietterich. 2000. Ensemble methods in machine learning. In *International Workshop on Multiple Classifier Systems*.

Jack FitzGerald, Christopher Hench, Charith Peris, Scott Mackie, Kay Rottmann, Ana Sanchez, Aaron Nash, Liam Urbach, Vishesh Kakarala, Richa Singh, Swetha Ranganath, Laurie Crist, Misha Britan, Wouter Leeuwis, Gokhan Tur, and Prem Natarajan. 2022. Massive: A 1m-example multilingual natural language understanding dataset with 51 typologically-diverse languages.

Sonal Gupta, Rushin Shah, Mrinal Mohit, Anuj Kumar, and Mike Lewis. 2018. Semantic parsing for task oriented dialog using hierarchical representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2787–2792, Brussels, Belgium. Association for Computational Linguistics.

L.K. Hansen and P. Salamon. 1990. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001.

William Held, Christopher Hidey, Fei Liu, Eric Zhu, Rahul Goel, Diyi Yang, and Rushin Shah. 2023. Damp: Doubly aligned multilingual parser for task-oriented dialogue.

Sophie Henning, William Beluch, Alexander Fraser, and Annemarie Friedrich. 2023. A survey of methods for addressing class imbalance in deep-learning based natural language processing. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 523–540, Dubrovnik, Croatia. 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 (Volume 1: Long Papers)*, pages 908–921, Online. Association for Computational Linguistics.

Christopher Hidey, Fei Liu, and Rahul Goel. 2022. Reducing model churn: Stable re-training of conversational agents. In *Proceedings of the 23rd Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 14–25, Edinburgh, UK. Association for Computational Linguistics.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network.

Heinrich Jiang, Harikrishna Narasimhan, Dara Bahri, Andrew Cotter, and Afshin Rostamizadeh. 2022. Churn reduction via distillation.

S. Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, Austin, Texas. Association for Computational Linguistics.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and scalable predictive uncertainty estimation using deep ensembles.

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

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization.

Mahdi Milani Fard, Quentin Cormier, Kevin Canini, and Maya Gupta. 2016. Launch and iterate: Reducing prediction churn. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

Rafael Müller, Simon Kornblith, and Geoffrey Hinton. 2020. When does label smoothing help?

Steven Reich, David Mueller, and Nicholas Andrews. 2020. Ensemble Distillation for Structured Prediction: Calibrated, Accurate, Fast—Choose Three. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5583–5595, Online. Association for Computational Linguistics.

Subendhu Rongali, Luca Soldaini, Emilio Monti, and Wael Hamza. 2020. *Don't Parse, Generate! A Sequence to Sequence Architecture for Task-Oriented Semantic Parsing*, page 2962–2968. Association for Computing Machinery, New York, NY, USA.

Gil I. Shamir and Lorenzo Coviello. 2020. Anti-distillation: Improving reproducibility of deep networks.

Gil I. Shamir, Dong Lin, and Lorenzo Coviello. 2020. Smooth activations and reproducibility in deep networks.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: The impact of student initialization on knowledge distillation. *CoRR*, abs/1908.08962.

A Ethics

All datasets used in this experiments are intended for research purposes only. We verified that the datasets do not contain personally identifiable information.

B Hyper-parameter Search and Settings

For our experiments, we used the TPU v2 via Google Cloud[9]. Table 8 displays the hyperparameter values used for our experiments. We use the relu activation function for our non-linearity and for our optimizer we use Adam with weight decay (Loshchilov and Hutter, 2019). The output vocabulary and bert embedding vocabulary is embedded into 128 dimensions as done in the pointer decoder setting Rongali et al. (2020). For all embedding layers (wordpiece, BERT, and output before softmax) we apply dropout.

C Additional Results

Negative Pairs To counter overtriggers, we explored adding negative example pairs. Negative examples have *different* function call signatures but do have overlapping spans. The spans occurs only in the one of the sequences in the pair. This setting was designed to force the model to generate dissimilar embeddings for the same span appearing in different contexts. However, we did not obtain additional improvement with this approach. The results can be seen in Table 9.

Hyperparameter/ Dataset / Setting	Value
Learning rate / - / Without CD	5e-6
Learning rate / - / CD	1e-5
Batch size / - / Without CD	32
Batch size / - / CD	128
Train Steps / TOP (Gupta et al., 2018) / Without CD	500k
Train Steps / TOPv2 (Chen et al., 2020) / Without CD	800k
Train Steps / MASSIVE (FitzGerald et al., 2022) / Without CD	800k
Train Steps / - / CD	272k
num decoder heads / - / -	8
num decoder layers / - / -	4
max decode length / - / -	51
Span loss weight(λ) / - / -	1
CD loss weight / - / CD	1
backoff smoothing(α) / TOP (Gupta et al., 2018) / -	0.25
backoff smoothing(α) / TOPv2 (Chen et al., 2020) / -	0.25
backoff smoothing(α) / MASSIVE (FitzGerald et al., 2022) / -	0.5

Table 8: Hyperparameter values across datasets. '-' Dataset represents that the value was same across all datasets. '-' setting represents that the value was same across all settings.

D Results by Function Call Signature

We present results grouped by function call signature in Tables 10 and 11. We compare the performance on "LS + Pairwise/JSD" to the LS baseline. We only include function call signatures that occur at least 5 times in the test set. Table 10 shows the most improved function call signatures. Many of these function call signatures include generic slots such as place names and date/times where the span similarity approach is likely to better cluster these spans in vector space when training the encoder. Conversely, we oberve degradation on some function call signatures (Table 11). Many of these function call signatures are argument-less intents as discussed in Section C.

[9]https://cloud.google.com/tpu

Model	TOP		TOPv2		MASSIVE	
	EM(@10)	AGR	EM(@10)	AGR	EM(@10)	AGR
Pairwise	**81.27 (72.83)**	78.95	84.23 (77.95)	83.81	66.79 (56.12)	**65.57**
Pairwise JSD	80.93 (72.77)	**79.04**	84.37 **(78.21)**	84.04	67 (56.36)	65.3
Pairwise JSD + Neg Pair	80.8 (71.90)	77.5	84.22 (77.6)	83.27	66.80 (55.65)	63.99
Pairwise JSD ($\alpha = 0.1$)	80.81 (72.13)	78.14	84.36 (78.12)	**84.13**	67.06 (56.05)	64.53
Pairwise JSD ($\alpha = 0.25$)	81.01 (72.48)	78.56	**84.39** (78.13)	84.01	**67.10** (56.25)	64.96
Pairwise JSD ($\alpha = 0.5$)	80.91 (72.27)	78.49	84.34 (78.04)	83.95	67.02 **(56.49)**	65.37

Table 9: Model performance (over N = 10 runs) when trained on **academic** datasets. EM: exact match (mean over 10 runs). EM@10: EM if all 10 models are correct. AGR: model agreement.

Function Call Signature	EM(@10)	AGR	EM@10 Delta
in:iot-hue-lightdim	100 (100)	100	31.25
in:social-post	42.5 (25)	25	25
in:audio-volume-down	93.334 (88.89)	88.89	22.22
in:calendar-set(sl:general-frequency=)	66 (40)	40	20
in:email-querycontact	46 (20)	40	20
in:transport-query	54 (40)	40	20
in:weather-query(sl:time=)	40 (20)	20	20
in:play-audiobook(sl:audiobook-name=)	30.002 (14.29)	42.86	14.29
in:iot-wemo-on(sl:device-type=)	76.25 (62.5)	62.5	12.5
in:recommendation-events(sl:place-name=)	71.114 (55.56)	66.67	11.12
in:calendar-query(sl:date=)	64.165 (41.67)	45.83	8.34
in:calendar-set(sl:event-name=,sl:time=)	25.386 (7.69)	15.38	7.69

Table 10: Most improved examples on MASSIVE relative to the LS baseline, grouped by function call signature and sorted by EM@10. Only function call signatures with at least 5 examples in the test set are presented.

Function Call Signature	EM(@10)	AGR	EM@10 Delta
in:alarmset	62 (40)	40	-40
in:recommendationevents	63.336 (33.33)	50	-33.34
in:transporttaxi(sl:transportagency=)	65.002 (50)	66.67	-33.33
in:listscreateoradd(sl:listname=)	62.353 (35.29)	41.18	-29.42
in:newsquery(sl:placename=)	54.284 (42.86)	57.14	-28.57
in:qacurrency(sl:currencyname=)	85 (37.5)	37.5	-25
in:playgame	66.669 (44.44)	55.56	-22.23
in:alarmset(sl:date=,sl:time=)	81.113 (55.56)	55.56	-22.22
in:alarmquery	91.737 (73.91)	73.91	-21.74
in:weatherquery	60 (55)	70	-20
in:socialquery(sl:mediatype=)	73 (40)	50	-20

Table 11: Most regressed examples on MASSIVE relative to the LS baseline, grouped by function call signature and sorted by EM@10. Only function call signatures with at least 5 examples in the test set are presented.