MIXTURE OF DOMAIN EXPERTS FOR LANGUAGE UNDERSTANDING: AN ANALYSIS OF MODULARITY, TASK PERFORMANCE, AND MEMORY TRADEOFFS

Benjamin Kleiner[*], *Jack G.M. Fitzgerald*[†], *Haidar Khan*[†], *Gohkan Tur*[†]

AWS AI Labs[*]
Amazon Alexa AI[†]

ABSTRACT

One of the limitations of large-scale machine learning models is that they are difficult to adjust after deployment without significant re-training costs. In this paper, we focus on NLU and the needs of virtual assistant systems to continually update themselves through time to support new functionality. Specifically, we consider the tasks of intent classification (IC) and slot filling (SF), which are fundamental to processing user interaction with virtual assistants. We studied six different architectures with varying degrees of modularity in order to gain insights into the performance implications of designing models for flexible updates through time. Our experiments on the SLURP dataset, modified to simulate the real-world experience of adding new intents over time, show that a single dense model yields 2.5 – 3.5 points of average improvement versus individual domain models, but suffers a median degradation of 0.4 – 1.1 points as the new intents are incorporated. We present a mixture-of-experts based hybrid system that performs within 2.1 points of the dense model in exact match accuracy while either improving median performance for untouched domains through time or only degrading by 0.1 points at worst.

1. INTRODUCTION

Virtual assistants, such as Amazon Alexa, Google Assistant, and Apple Siri, are software systems that recognize speech, use Natural Language Understanding (NLU) models to understand the speech, and then perform tasks based on the interpretations from the NLU models, including setting timers, making calls, and playing music [1, 2, 3]. In its most foundational form, an NLU model must parse the users intent (EX: "play music"), a task called Intent Classification (IC), and it must also fill applicable slots (EX: determine that "stairway to heaven" is a song title), a task called Slot Filling (SF) [4, 5]. Suppose there is a system in which developers are organized into domain-based foci, where a domain is a set of one or more intents and corresponding slots. Unlike academic setups in which the annotation schema is predefined, the set of domains, intents, and slots in a real-world system is dynamic and changes regularly. A given developer should be able to add or modify a new intent or domain without retraining the full system and without degrading the task performance of other domains. Our contributions include:

- The first use of a mixture-of-experts architecture for NLU (to our knowledge),
- Two additional novel architectures, including a hybrid mixture-of-experts architecture and one with implicit domain classification, and
- Tradeoff analyses across domain modularity, task performance, and memory.

2. DATASET

We use the SLURP [6] dataset for our experiments and make the following modifications for our use case. First, we define the domain of an utterance to be the "scenario" field in SLURP. Next, we define the intent to be the scenario-action pair of an utterance. For example, if the scenario is "email" and the action is "addcontact", then we label the intent "email-addcontact". This results in 18 domains and 60 unique intents. We leave the slot labels as is, which yields a unique slot count of 55. SLURP is composed of 12k training examples, 2k development examples, and 3k test examples.

In order to simulate the real-world requirements of an adaptive NLU system, we split the data into 5 timesteps.

Corresponding author: **kleinerb@amazon.com**

978-1-6654-7189-3/22/$31.00

At $t = 0$, we hold out some domains and intents and then incrementally add them back in at each subsequent timestep. In order to cover several different real-world scenarios, we add a varying portion of the data at each timestep. Specifically, we use the schedule described in Table 1.

Timestep (t)	Domains	Intents
0	17	0
1	0	1
2	1	n
3	0	2
4	0	2

Table 1. The number of domains or intents added to the system at each timestep. n is the number of intents contained in the domain being added at $t = 2$.

Adding intents is more common than adding new domains, so in 3 timesteps we add new intents to existing domains and in one timestep we added a new domain. Domains and intents were chosen at random, and we conducted 4 different trials on different domain/intent arrangements, later averaging results across all trials.

The dataset was designed to mimic a major-minor update paradigm in which all domain components are retrained during major version changes, whereas a given domain developer can update only their domain in a minor version update. Thus $t = 0$ corresponds to a major version update, whereas $t = 1, 2, 3, 4$ are minor updates.

3. ARCHITECTURES

We consider six different system architectures that can handle NLU in multiple domains. Below, we describe each model separately.

Dense is an encoder followed by sequence and token classification heads. The Dense model sees data from all domains and its output space is the full output space (all intent and slot labels). The model is entirely retrained at each timestep on all of the available data at that time.

Individual is a collection of individual Dense models for each domain, whose predictions are aggregated by a domain classifier. Each individual model's output space is restricted to its own domain, and it only sees data from its own domain at training time. When an intent is added, we restart training again for that domain model from the original pretrained model weights.

Shared Utterance Encoder (SUE) is a shared encoder with separate heads for each domain and task. At $t = 0$, we train both the encoder and the classification heads. For $t > 0$, we freeze the encoder and only retrain heads for domains in which a new intent is added. New domains also use the same frozen encoder.

Mixture Of Experts (MOE) is a mixture of experts style model [7] where every feed-forward network in the transformer layers is replaced by an MoE layer. Each expert in the MoE layer corresponds to a particular domain, following the modular system introduced by DEMix Layers [8]. Additionally, there is a separate classification head per domain and task.

Hybrid is a dense encoder for the first 6 layers followed by domain MoE layers for the last 6 layers.

Prototype is the same as Hybrid except that it uses domain "prototypes" to perform expert routing at inference time. This method is described in more detail in Section 4.2.

The base architecture for all systems is the pretrained XLM-Roberta [9] [10] encoder from the HuggingFace library [11]. We then add a sequence classification head for the IC task and a token classification head for the SF task in the style of [12]. All models are trained using a joint loss function which is the sum of cross entropy loss for both IC and SF. To perform test-time routing among domain-specific components for Individual, SUE, MOE, and Hybrid systems, we use a separate domain classifier (DC) model whose details are given in Section 4.1. Additionally, each system other than Dense and Individual make use of an "Out of Domain" label to help recover from domain routing errors, as described in Section 4.3. We summarize all of these attributes in Table 2.

Arch	Dense	MOE	OOD	DC	Freeze
Dense	12	0	No	No	No
SUE	12	0	Yes	Yes	Yes
Hybrid	6	6	Yes	Yes	Yes
Proto	6	6	Yes	No	Yes
MOE	0	12	Yes	Yes	Yes
Individual	12	0	No	Yes	N/A

Table 2. Attributes of each architecture. Dense and MOE denote number of encoder layers of respective type. OOD, DC, and Freeze indicate whether "Out of Domain" labels, domain classifier model, and shared parameter freezing are used.

The training behavior for all systems across timesteps is given by the following three rules.

- The embeddings are always frozen.
- At $t = 0$, all encoder and head parameters are allowed to train.
- For $t > 0$, any shared parameters are frozen and any domain-specific components for which there was no change in intent are frozen, in keeping with "minor version" updates from $t = 1$ to $t = 4$.

At inference time, all domain components are run, as the correct domain is unknown. This is similar to the setup in DEMix [8], except that we use different inference time routing methods as described in Section 4.

4. DOMAIN CLASSIFICATION AND ROUTING

Domain classification and routing are key aspects of a modular NLU system. In order to effectively achieve independence of the domain-specific components, we must be able to determine how to route incoming data among these components. Ultimately, the domain-specific modules must work together as one system at inference time, and their interaction is critical to task performance.

We explored several techniques for routing between domain modules with the goal of aiding task performance subject to the constraint of training independence.

4.1. Domain Classifier

The primary method by which we perform DC in our experiments is to use a separate XLM-Roberta encoder model trained specifically for domain classification. We use the probabilities output from this model as mixing weights for our MoE style models. We found that the effect of using the probabilities as expert weights rather than taking the argmax was negligible, but we discuss this further in Section 4.4. The distinct DC model is the most accurate for domain classification, but the system incurs a non-trivial training cost at each timestep, which we discuss further in Section 5.1. Inference latency can be made negligible at the expense of memory if desired by replicating the necessary components of the IC/SF models and running the DC model in parallel.

4.2. Clustering

Alternatively, we explored unsupervised methods in an effort to maintain modularity while also reducing parameter count and training cost relative to the encoder-based DC model. [13] show that Gaussian Mixture Models can achieve reasonably accurate (nearly 90% in the best case) domain classification results on a multi-domain text corpus using the hidden representations produced by a pretrained BERT encoder. We investigated this on our task and also looked at the intermediate representations rather than just the final one. Consistent with prior work, we use the mean of all token representations for each sequence. Additionally, we compared this approach with the few-shot classification technique proposed by [14] for Prototypical Networks. This algorithm computes a prototype vector for each class from a small support set and compares the squared Euclidean distance between test data representations and the prototype vectors of each class, assigning the data to the nearest class. In our case

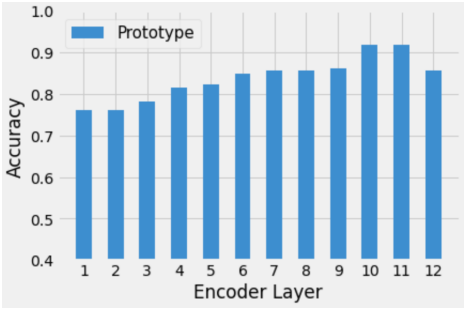


Fig. 1. Domain Classification Accuracy achieved using the mean-pooled hidden representations produced at each layer of the Encoder. This example was computed on a single t0 trial with 17 domains using the fine-tuned Dense model. We observe similar accuracy for the Hybrid model in the first 6 layers as well

we have much more data to use, but we find the method attractive from compute, memory, and modularity perspectives. As shown in Figure 1, the Prototype method is able to achieve approximately 85% domain classification accuracy beginning halfway through the encoder layers. Ultimately, we chose to use the prototype algorithm due to its simplicity and comparable or better performance, particularly in earlier layers. Using class prototypes to leverage the hidden representations already present in the encoder is extremely compute efficient as we see in the training cost comparison in Section 5.1, maintains training independence across domains since it is decoupled from the encoder, but suffers in overall accuracy as shown in Figure 3.

4.3. "Out Of Domain" Labels

In order to maintain independence of domain-specific classification heads, we restrict their output spaces to the labels within their own domain. We can freely add other domains or intents to the system without having to update any of the domain-specific modules in this way. Additionally, we can add an "out of domain" label to each intent classifier to help us recover from a domain classification error. The method works as follows. We augment the training set by duplicating examples, randomly switching their domain label to another domain, and switching their intent label to "out of domain". We keep the distribution of out of domain examples consistent for each domain and we experimented with a few varying amounts. In the end, we chose to use an amount of "out of domain" examples equal to the original data amount for each domain. In the final training set, 50% of each domain's training data comes from another domain and is given an "out of domain" intent label. At inference time, if the predicted domain-specific intent classifier returns an "out of domain" prediction, we assume we have a routing error and try the next-best domain.

t	Without	With OOD
0	0.703	0.718
1	0.712	0.720
2	0.710	0.717
3	0.708	0.722
4	0.706	0.714

Table 3. Exact Match Accuracy for the Hybrid model with and without using the OOD label for one time trial.

We experimented with simply iterating over domain-specific output heads and alternatively, reverting to the earliest MoE layer and running through all subsequent layers again with the next best domain selected. We found the latter approach to be slightly more effective in terms of task performance. One consideration with this step though is that in the worst case, inference computational complexity moves up to $\mathcal{O}(d)$ for d domains. That could also be mitigated by creating copies of each MOE layer and running them in parallel, but at the expense of memory, of course.

4.4. Mixing

Mixing techniques were mostly ineffective in the paradigm where we wanted to maintain strict training decoupling across domains. Using the classification probabilities from the methods described above as mixing weights for the domain modules produced negligible difference in results from those obtained using only the top-1 component. This is due to the fact that the probabilities are generally heavily concentrated on the top domain (median weight for the top domain is $.985$), so there is little weight going to the other domains in most examples. Separately, we tried learning a mixing function in an additional training step after the models had been fine-tuned. We froze the models entirely aside from our additional mixing modules in order to maintain the desired training independence. However, these adapter-like modules were not able to train in this setup. [15] also noted the inability of adapter networks to train from random initialization on an already trained network. They found that initializing to a near-identity function helped the modules train, but this approach is not directly applicable to our task. Our setup is related, but an important difference is that we are learning a routing function as opposed to adapting the already existing computational graph.

5. RESULTS

5.1. Training Compute Comparison

All models are identical in terms of Floating Point Operations (FLOPs) per training forward pass, not including the DC component. The models that make use of MoE layers are sparsely activated at training time. Since each IC/SF component is FLOP matched, the variation in required computing resources comes primarily from the difference in the amount of training data used and whether or not an additional domain classifier module is needed.

Each system except for Dense and Prototype use a separate Domain Classifier, which uses the same XLM-Roberta architecture as the rest of the models. At each timestep this DC model runs on the full training set. The Prototype architecture is the most compute efficient of those we study, since it does not require a DC model and the prototype computation requires only a single forward pass through the training set at each timestep.

For all models other than Dense, we only require a subset of the full training set corresponding to the domains that involved a change, so we observe a steep decrease in training cost after $t = 0$. Training com-

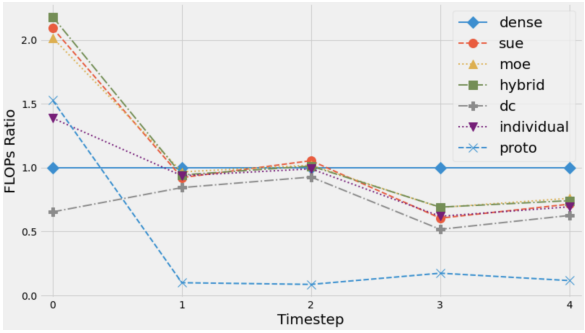


Fig. 2. Observed ratio of training FLOPs required vs Dense, averaged over 4 variations of time splits. Each model is fine-tuned until validation improvement ceases for 10 epochs.

Architecture	Memory Complexity	Ratio
Dense	$\mathcal{O}(e+c)$	1
SUE	$\mathcal{O}(2e+cd)$	2
Proto	$\mathcal{O}(e(1+\frac{1}{3}d)+cd)$	7
Hybrid	$\mathcal{O}(e(2+\frac{1}{3}d)+cd)$	8
MOE	$\mathcal{O}(d(\frac{2}{3}e+c))$	14
Individual	$\mathcal{O}(d(e+c))$	19

Table 4. Inference memory usage across architectures. Ratio is the approximate ratio of memory usage compared to the Dense model. Complexity is given in terms of a single encoder e, single classification head c, and the number of domains d.

pute at subsequent timesteps for these models is driven primarily by the domain classifier component. For the models which make use of the "Out of Domain" label (explained in Section 4.3), we also use an equivalent amount of randomly sampled data from other domains, so the requirement is double what it would be otherwise.

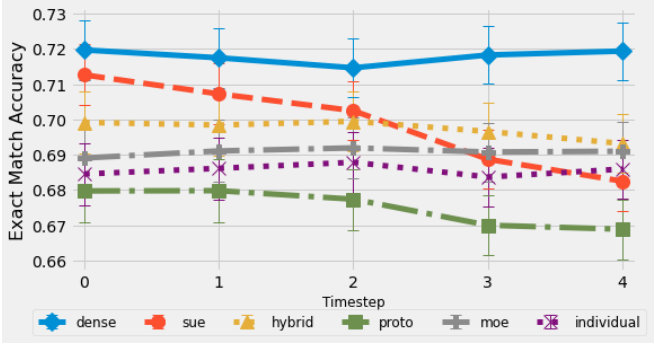
5.2. Memory Comparison

All of the architectures we experiment with use inference memory in proportion to the number of domains involved, aside from the Dense model. See Table 4.

5.3. Task Performance

The metric we focus on for our combined task performance is exact match accuracy. We consider a prediction to be an exact match if all tokens are classified correctly in slot labeling and the intent is also correctly classified.

We observe a positive correlation between the number of dense layers and task performance at $t = 0$ (see Figure 3). Greater connectedness and more diverse data flowing to a greater percentage of parameters seems to allow more knowledge sharing across domains, driving exact match accuracy to 72.0% for Dense versus 68.5% for Individual. Additionally, we observe that systems with a greater proportion of frozen parameters perform worse for newly added intents at later timesteps. The SUE model drops 3 points from $t = 0$ to $t = 4$, as 99% of its parameters are frozen after the initial training period. The Individual model, which has no frozen parameters, shows little change in performance over time in contrast, while the Hybrid model, which has 66% of its parameters frozen starting at $t = 1$, degrades 0.6 points.



system	$t=0$ (%)	$t=1$ (%)	$t=2$ (%)	$t=3$ (%)	$t=4$ (%)
dense	**72.0 ± 0.9**	**71.8 ± 0.9**	**71.5 ± 0.8**	**71.8 ± 0.8**	**71.9 ± 0.8**
sue	71.3 ± 0.9	70.7 ± 0.9	70.3 ± 0.8	68.9 ± 0.8	68.2 ± 0.8
hybrid	69.9 ± 0.9	69.8 ± 0.9	69.9 ± 0.8	69.7 ± 0.8	69.3 ± 0.8
proto	68.0 ± 0.9	68.0 ± 0.9	67.7 ± 0.9	67.0 ± 0.9	66.9 ± 0.8
moe	68.9 ± 0.9	69.1 ± 0.9	69.2 ± 0.9	69.1 ± 0.8	69.1 ± 0.8
individual	68.5 ± 0.9	68.6 ± 0.9	68.8 ± 0.9	68.4 ± 0.8	68.6 ± 0.8

Fig. 3. Exact Match Accuracy across timesteps for each system, averaged across 4 variations of timestep data construction. Error bars indicate 95% confidence interval, which is between $\pm 0.8\%$ and $\pm 0.9\%$ (absolute) for all systems and timesteps.

5.4. Modularity

We compare the degree of modularity in the systems in two ways. First, we examine the task performance change over time on unmodified domains (no intents added over time). Figure 4 shows that greater variability is introduced for unmodified domains for the Dense model, which is entirely unified across domains and does not freeze any parameters. There is still some variability for the other models since they rely on a domain classifier component which is retrained each timestep, but they are better able to preserve existing performance overall.

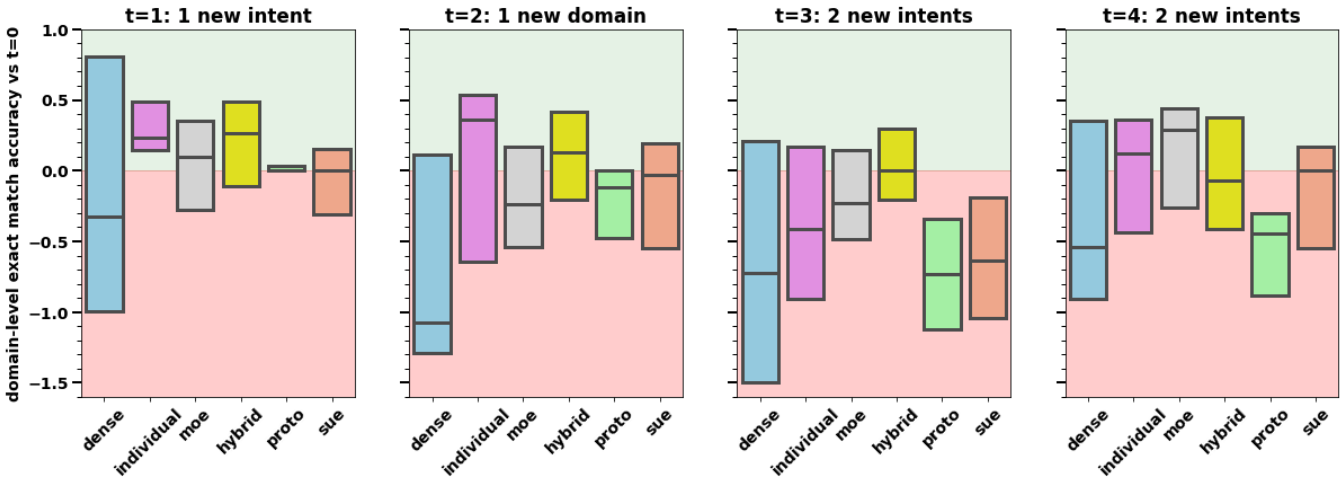


Fig. 4. Q1, median, and Q3 absolute change in domain-averaged exact match accuracy versus $t = 0$ for untouched domains only. Results are averaged across 4 variations of timestep data construction.

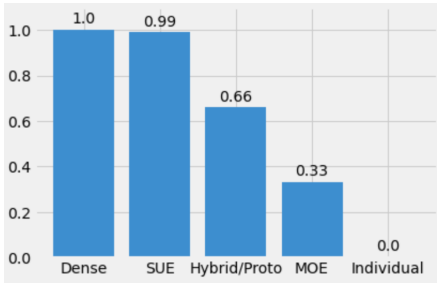


Fig. 5. Fraction of parameters for each domain that are shared with other domains (excluding embeddings and DC component)

The InterQuartile Range (IQR) of domain-aggregated exact match accuracy ranges from 1.3 to 1.8 for Dense, whereas the IQR for Hybrid, for example, ranges from 0.50 to 0.78. In Figure 4, both proximity to 0 and a tight distribution correspond to domain performance stability, which in turn means less operational burden. MOE and Hybrid both exhibit relatively tight distributions and small amounts of domain degradation, both sometimes outperforming Individual. Additionally, we consider the percentage of total parameters that are shared across domains as a measure of domain coupling, shown in Figure 5.

6. CONCLUSION

The Dense system outperforms all others in most dimensions, but it comes with a cost of decreased modularity, and thus increasing operational burden as more domain-focused developers work on the system together. Considering the other systems across all dimensions, we find the Hybrid and MOE systems to be most compelling. On exact match performance, the Hybrid system begins with a 1 point advantage over MOE at $t = 0$, but this gradually declines by $t = 4$ to comparable performance (69.3% vs 69.1%). The percentage of shared parameters is 66% for Hybrid vs 33% for MOE, but we do not observe a proportionate difference in existing domain performance. Specifically, the MOE model median change for existing domains at $t = 4$ is 0.3 points higher than Hybrid, but it is only ahead by 0.1 points in the first and third quartiles. From a computation perspective, MOE and Hybrid consume nearly identical training FLOPs, though the Hybrid model uses roughly 60% of the number of parameters and inference memory as the MOE model.

In conclusion, we have considered a paradigm in which minor, domain-specific model changes are performed between less frequent major system updates. In this paradigm, the Hybrid model offers an effective way to decouple the domains within the NLU model, achieving better exact match performance, memory usage, and equal training compute with only a slight degradation in existing domain performance relative to all but the Dense model. The MOE model is a close contender, and it may perform equally well as the number of minor updates increases. We leave such questions, as well as experimentation on different datasets and more exhaustive hyperparameter tuning, to future work.

7. REFERENCES

[1] Steve J. Young, "Talking to machines (statistically speaking)," in *INTERSPEECH*, 2002.

[2] Ye-Yi Wang, Li Deng, and Alex Acero, "Spoken language understanding," *IEEE Signal Processing Magazine*, vol. 22, pp. 16–31, 2005.

[3] Gokhan Tur and Renato De Mori, "Spoken language understanding: Systems for extracting semantic information from speech," 2011.

[4] Charles T. Hemphill, John J. Godfrey, and George R. Doddington, "The ATIS spoken language systems pilot corpus," in *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*, 1990.

[5] Shikib Mehri, Mihail Eric, and Dilek Hakkani-Tur, "Dialoglue: A natural language understanding benchmark for task-oriented dialogue," 2020.

[6] Emanuele Bastianelli, Andrea Vanzo, Pawel Swietojanski, and Verena Rieser, "Slurp: A spoken language understanding resource package," 2020.

[7] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," 2017.

[8] Suchin Gururangan, Mike Lewis, Ari Holtzman, Noah A. Smith, and Luke Zettlemoyer, "Demix layers: Disentangling domains for modular language modeling," 2021.

[9] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019.

[10] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov, "Unsupervised cross-lingual representation learning at scale," 2020.

[11] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online, Oct. 2020, pp. 38–45, Association for Computational Linguistics.

[12] Qian Chen, Zhu Zhuo, and Wen Wang, "BERT for joint intent classification and slot filling," *CoRR*, vol. abs/1902.10909, 2019.

[13] Roee Aharoni and Yoav Goldberg, "Unsupervised domain clusters in pretrained language models," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online, July 2020, pp. 7747–7763, Association for Computational Linguistics.

[14] Jake Snell, Kevin Swersky, and Richard S. Zemel, "Prototypical networks for few-shot learning," 2017.

[15] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly, "Parameter-efficient transfer learning for NLP," in *Proceedings of the 36th International Conference on Machine Learning*, Kamalika Chaudhuri and Ruslan Salakhutdinov, Eds. 09–15 Jun 2019, vol. 97 of *Proceedings of Machine Learning Research*, pp. 2790–2799, PMLR.