# Selective Progressive Prompts: Improve Forward Transfer by Selectively Concatenating Prompts from Prior Tasks in Continual Learning

Ty Feng, Raj Kunamaneni, Srivatsan Srikanth, Henry Chou

University of California, Davis

**Abstract**

Continual learning in natural language processing often struggles with the integration of knowledge from multiple tasks in a task sequence. Existing methods either learn a separate per-task prompt, neglecting the potential benefits of forward transfer, or indiscriminately concatenate all the previously learned prompts, compromising accuracy when learning a sequence of heterogeneous tasks. In this paper, we introduce Selective Progressive Prompts, a modified version of Progressive Prompts for continual learning. Instead of concatenating every prompt from prior tasks, we perform a cosine similarity check between each input embedding and the input embeddings from prior tasks. If there is at least one prior input embedding similar enough to the current input embedding, we would use the previously learned prompts when learning the current task. Otherwise, we would learn a separate per-task prompt using prompt tuning. Experiments on Forward Transfer benchmarks showed a higher accuracy using our approach than Progressive Prompts in five out of the six experiments on pairs of dissimilar NLP tasks. Our approach achieved the highest accuracy when compared to Prompt Tuning and Progressive Prompts for three of the six task pairs. Our approach enables selective integration of knowledge when learning tasks sequentially in continual learning, paving the way for applications requiring dynamic, long-term language understanding. Our code base can be found at `https://github.com/ytyfeng/SelectiveProgressivePrompts`.

## 1 Introduction

Determining ways to learn a task sequence while allowing previously learned tasks to be retained remains an important goal in Continual Learning (CL). In CL, an ideal language model should be able to learn from multiple sequential sequences of tasks in a manner that prevents catastrophic forgetting and allows forward transfer to occur. This means that the model should not only learn the knowledge pertaining to new tasks but also leverage its previously learned knowledge from older tasks to improve its learning performance on the new task.

Many existing language models only excel in single-task learning. Progressive Networks [5] try to solve the problem of forgetting previously learned tasks and allow for forward transfer but they add a new model for each new task, making it expensive to train. More recently, Progressive Prompts [4] have been used to support forward transfer while retaining knowledge of previous tasks via Progressive Networks and Prompt Tuning [2] for each task. It concatenates previously learned prompts for prior tasks to the current task prompt using the progressive network, thereby leveraging prior knowledge from previous tasks to help learn future tasks.

While Progressive Prompting performs prompt concatenation, it does not restrict the prompts that are being added to the network. All of the previous prompts are concatenated at each sequential step, regardless of the input task text similarity. Consequently, the previous prompts from irrelevant tasks could interfere with the current prompt, compromising the accuracy when learning the current task. Therefore, when learning a sequence of heterogeneous tasks, particularly in the real-world setting, Progressive Prompts may not perform as well as when learning a sequence of similar tasks.

In order to avoid concatenating irrelevant prompts, we propose a modified method, Selective Progressive Prompts, utilizing text similarity. For each task text in the training set, we calculate the similarity of the text with the texts of all the previous task texts and concatenate the corresponding prompts if there is a high similarity.

This modification of the progressive prompts will ensure that only prompts with similar texts are concatenated to the current prompt. This selection mechanism will potentially improve the forward transfer in continual learning, as only previously learned prompts from similar tasks will be considered during forward transfer.

In this paper, we will cover the methodologies used in calculating the similarities in the prompt texts. To study the effect on forward transfer between different tasks, we will compare our approach with progressive prompts [4] and original prompt tuning [2] on learning 6 different pairs of NLP tasks. Finally, we will discuss what the results signify regarding our selective progressive prompt tuning method.

## 2 Background

### 2.1 Model Fine-tuning

In model fine-tuning, the internal language model parameters such as weights are being adjusted such that a general-purpose language model can be specialized into one that better performs on a specific set of tasks. In the context of continual learning and progressive networks, the main drawback of using model fine-tuning is that we need to train multiple separate models when learning a sequence of tasks [4, 9]. Model fine tuning requires all model parameters to be updated, and requires storing a separate tuned model for each of the downstream tasks. Currently, available language models each have a large number of parameters. For instance, the baseline BERT model has 220 million parameters, while both the GPT-3 and LLaMA language models each have billions of parameters that need to be tuned, each requiring multiple large memory capacity GPUs to load and train the model [3, 6]. Thus, tuning a large language model is resource intensive, while a stack of these fine-tuned large language models in a progressive network will require a huge amount of computational resources and time to finish model tuning.

In model fine-tuning, we maximize the log-likelihood of the output $y$ over the class probability distribution $p_\theta$ by performing gradient updates, given a task $T$ and input text $x$, by tuning model parameters $\theta$ [4]:

$$\max_{\theta} \sum_{x,y \in T} \log p_\theta(y|x) \tag{1}$$

### 2.2 Prompt Engineering

The basic premise of prompt tuning comes from an area of prompt design or prompt engineering, where a frozen language model's behavior could be conditioned through carefully crafted text prompts serving as input into the model. Freezing a model's parameters while still modifying its behavior is necessary for many downstream NLP tasks. However, prompt engineering requires manual labor to design and deploy, and it is challenging to know if handcrafted prompts can achieve optimal performance on a given task. Additionally, language models typically have a limited context window, and only a limited amount of conditioning text prompts can fit into the model as input. Therefore, prompt engineering is often limited in task performance compared to prompt tuning methods that are trained on data [2].

## 2.3 Prompt Tuning

Rather than manually designing prompts through trial and error or relegating back to model fine-tuning, prompt tuning learns a $soft$ prompt that can best condition the language model for a specific task. With the language model parameters frozen, we train additional prompt parameters to learn a soft prompt that best conditions the language model for a specific task [2]. Since soft prompts are vectors in the embedding space, they can be optimized through gradient descent and backpropagation, which is how the best soft prompt is found. With the learned soft prompt prepended to the input task text, the frozen language model can more effectively perform certain tasks. The tuning on soft prompts removes the need to update all the parameters of a language model which is expensive and resource-intensive. Therefore, by adding a learned soft prompt to the input text, the given language model could use this to maximize the likelihood of obtaining the correct class label or prediction while keeping the language model's parameters frozen [2].

The learning objective of prompt tuning is therefore maximizing the log-likelihood of correct prediction $y$ given a learned soft prompt $P$ concatenated with input text $x$ for a task $T$, with the model parameters $\theta$ frozen and prompt parameters $\theta_P$ tunable:

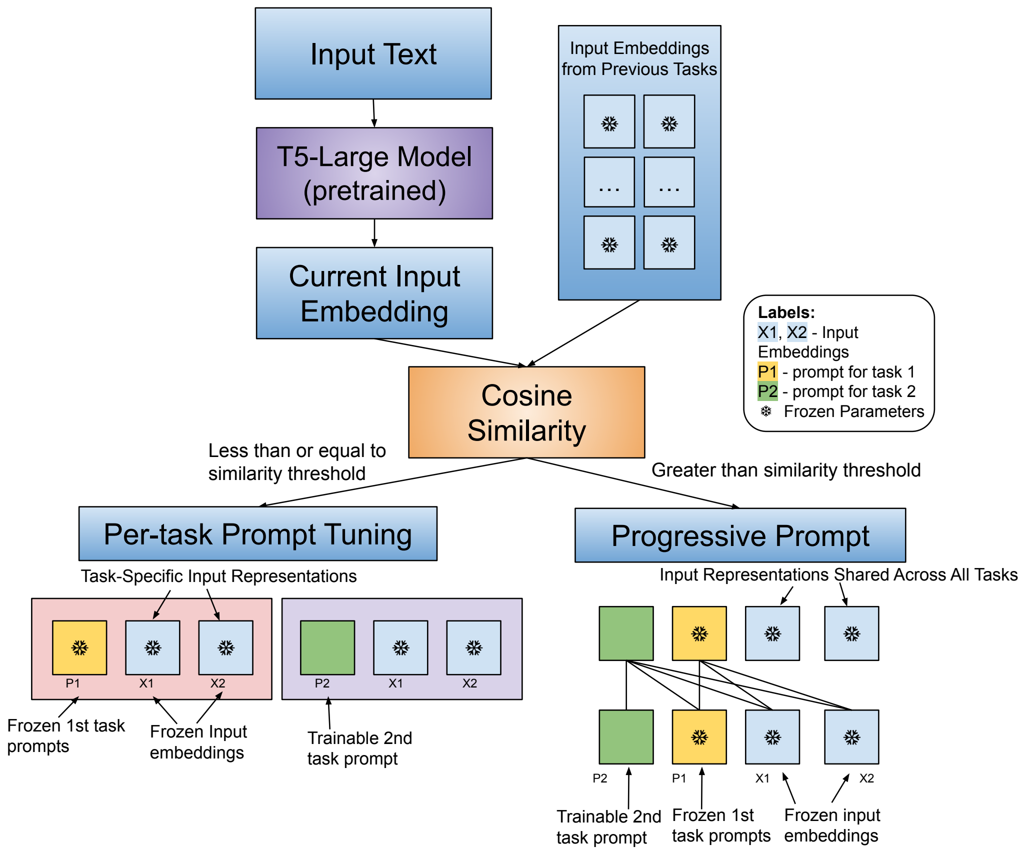
$$\max_{\theta_P} \sum_{x,y \in T} \log p_{\theta,\theta_P}(y|[P;x]) \tag{2}$$

## 2.4 Progressive Prompts

Progressive Prompts build upon the idea and benefits of prompt tuning to address the issue of forgetting. While Progressive Networks solve the issue of catastrophic forgetting while allowing forward transfer of prompts, such networks are expensive to run because they add a new model for every new task [4]. Progressive Prompts are able to remember the previously learned prompts and generalize their acquired knowledge to new tasks that they see by only learning a fixed number of prompt tokens for every task which uses significantly less memory than model fine-tuning [4]. By concatenating previously learned prompts to the current prompt, Progressive Prompt models can efficiently learn new prompts on top of the ones already learned, leveraging knowledge from previously learned tasks to learn subsequent tasks.

The learning objective of progressive prompts for a sequence of $k$ tasks $T$ is to tune prompt parameters $\theta_{P_k}$ for the current task that minimizes the negative log-likelihood of prediction $y$ given the previously learned soft prompts $P_k, \ldots, P_1$ concatenated with the input text $x$, with the model parameters $\theta$ and previous prompt parameters $\theta_{P_1}, \ldots, \theta_{P_k-1}$ remain frozen:

$$L(\theta_{P_k}) = -\sum_{x,y \in T_k} \log p(y|[P_k, \ldots, P_1, x], \theta, \theta_{P_1}, \ldots, \theta_{P_k}) \tag{3}$$



Input Text
T5-Large Model (pretrained)
Current Input Embedding
Input Embeddings from Previous Tasks
Labels:
X1, X2 - Input Embeddings
P1 - prompt for task 1
P2 - prompt for task 2
❆ Frozen Parameters
Cosine Similarity
Less than or equal to similarity threshold
Greater than similarity threshold
Per-task Prompt Tuning
Progressive Prompt
Task-Specific Input Representations
Input Representations Shared Across All Tasks
P1
X1
X2
P2
Frozen 1st task prompts
Frozen Input embeddings
Trainable 2nd task prompt
Trainable 2nd task prompt
Frozen 1st task prompts
Frozen input embeddings

Figure 1: A visual illustration of our Selective Progressive Prompts architecture. Given the input text, we generate an input embedding using the T5-Large Model. The input embedding will be used to calculate Cosine Similarity with the Previous Task Input Embedding. The similarity score would then be compared to the similarity threshold to either train a per-task prompt using Prompt Tuning or concatenate relevant prompts using Progressive Prompts.

## 3 Methods

### 3.1 Selective Progressive Prompts

Although the Progressive Prompts model is able to perform forward transfer among a sequence of tasks, it does not consider the case when the current task has little similarity to previous tasks. When training on a sequence of tasks that have little similarity, Progressive Prompts could lead to irrelevant prompts being added to the collective progressive prompts, which could negatively interfere with learning the current task. Our proposed method, Selective Progressive Prompting, aims to fix this issue. Selective Progressive Prompting modifies the original prompt concatenation mechanism in Progressive Prompts by implementing a selection criteria by which the model selects the task prompts that should be concatenated, instead of concatenating all the previous prompts to the current prompt.

Our training objective is therefore to find the prompt parameters $\theta_{P_k}$ that minimizes the negative log likelihood of $y$ given the input text $x$ concatenated with previously learned soft prompts $[P_k^*, \ldots, P_1^*]$ for previous tasks $T_k$, $(k \in 1...m)$ with model parameters $\theta$ kept frozen:

$$L(\theta_{P_k}) = -\sum_{\substack{x,y \in T_k \\ sim(x, X_{prev}) \geq \text{similarity\_threshold}}} \log p(y | [P_k^*, \ldots, P_1^*, x], \theta, \theta_{P_1}, \ldots, \theta_{P_k}) \tag{4}$$

where $P_i^*$ are the selectively chosen prompts that meet the input text similarity threshold out of all the available prior prompts $P_i$, and $sim(x, X_{prev})$ is the cosine similarity function comparing the current input embedding $x$ to the set of previous input embeddings $X_{prev}$. The intuition behind our approach is that there is a higher likelihood for the language model to generate correct results given prompts learned on only relevant tasks, rather than concatenating prompts from all prior tasks indiscriminately like in Progressive Prompts.

Figure 1 is a visual illustration of our Selective Progressive Prompts approach.

## 3.2 Text to Embeddings

To create embedding vectors, we used the pre-trained T5-large language model [3] to create embedding vectors from the input text, which are then used to perform the next steps of our Selective Progressive Prompts model. Specifically, for each input sequence, we used the pre-trained encoder of T5 to create a matrix of shape $SequenceLength$ by $EmbeddingVectorSize$. In our case, the sequence length is 512, meaning each sequence has a fixed number of tokens of 512, and the embedding vector size is 1024. We then performed mean pooling on the 512 per-token embedding vectors to create a single embedding vector for the entire input sequence of tokens, which is a 1D embedding vector of size 1024.

## 3.3 Input Tasks Similarity Calculation

Our Selective Progressive Prompt method hinges on the ability to gauge the semantic proximity of various input task texts. To achieve this, we implement a strategy that quantifies similarity through cosine similarity measures. After converting input texts into embedding vectors, we can calculate the closeness of any new task's representation to those of prior tasks.

Cosine similarity is an effective method for determining text similarity because it evaluates the cosine of the angle between two vectors — effectively capturing the orientation of texts in the vector space, rather than their magnitude. The resultant score, which spans the range of -1.0 (indicating completely dissimilar vectors) to 1.0 (indicating identical vectors), thus serves as a reliable indicator of textual similarity. When the cosine similarity score approaches 1.0, it highlights a high degree of similarity between the task embeddings, while a score near -1.0 suggests a strong dissimilarity.

The calculation of cosine similarity is given by the following mathematical expression:

$$\text{cosine similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Here, $\mathbf{A}$ and $\mathbf{B}$ are the vector representations of two task texts. The dot product of $\mathbf{A}$ and $\mathbf{B}$, denoted as $\mathbf{A} \cdot \mathbf{B}$, gauges the inherent directional consistency between the two vectors. The normalization by the product of the vectors' magnitudes, $\|\mathbf{A}\|$ and $\|\mathbf{B}\|$, serves to standardize the similarity score regardless of the text length or vector scale.

Using cosine similarity, we can check whether the current task is similar to previous tasks, which is how we determine whether we concatenate prompts from those tasks to the current prompt. As a result, we can ensure the relevance of previous prompts when learning the current task to yield better performance on a sequence of tasks in continual learning.

### 3.4 Similarity Threshold Comparison

In our Selective Progressive Prompt method, we determine the applicability of previously learned prompts to a current task by comparing the cosine similarity scores. Specifically, we calculate these scores between the current task's text embedding and those of previous tasks. If there is at least one previous task text with a similarity score that surpasses our predetermined similarity threshold, then it signals that the current task is sufficiently related to a previous one, so we can concatenate the previously learned prompts with the current prompt to promote forward transfer, similar to progressive prompts [4]. Otherwise, we would learn a separate per-task prompt using prompt tuning [2]. This selection mechanism allows us to selectively concatenate prompts based on input task similarities. The similarity threshold is a hyper-parameter that can be adjusted through a command line argument before training.

## 4 Experiment

### 4.1 Implementation Details

Our selection of the encoder-decoder T5 language model [3] for our project was driven by its superior generative capabilities, particularly when compared to encoder-only models like BERT [1], enabling versatile application across a broad spectrum of NLP tasks. We use the pre-trained T5-large model for all our experiments. We followed Progressive Prompt's [4] T5 implementation for our work, and we added our modified version of Progressive Prompts on top of their codebase. To train the continual learning model for learning soft prompts, we used 3 NVIDIA V100 GPUs from the Google Cloud Platform. Due to time constraints, we trained 10 epochs for each task. The time it took to finish running our experiments was around 3 days.

### 4.2 Datasets

We used the datasets from the Progressive Prompts for Continual Learning from (Zhang et al.,2015) [10], GLUE [8], and SuperGLUE [7] benchmarks, and IMDB [4]. Table 1 shows datasets used for the Forward Transfer experiments. GLUE (General Language Understanding Evaluation) and SuperGLUE are benchmarks that evaluate the performance of models on a diverse set of language understanding tasks, including tasks like sentiment analysis and textual entailment [8].

### 4.3 Experimental Setup

We ran six experiments on six different pairs of NLP tasks to study the *forward transfer* phenomenon. Forward transfer is when a task prompt learns the current task by exploiting knowledge learned from previous tasks. It is therefore crucial that the current task and previous tasks have high similarity with each other. When learning a sequence of tasks in continual learning, especially in a real-world setting, there is often no guarantee that the tasks are similar. We hypothesized that Progressive Prompts' performance would suffer when learning a pair of dissimilar tasks because the

| Dataset | Category | Task |
|---|---|---|
| AMAZON | CL Benchmark | Sentiment Analysis |
| SST2 | GLUE | Sentiment Analysis |
| QQP | GLUE | Paraphrase Detection |
| MRPC | GLUE | Paraphrase Detection |
| YELP | CL Benchmark | Sentiment Analysis |
| COPA | SuperGLUE | Questions and Answers |
| IMDB | Other | Sentiment Analysis |
| WiC | SuperGLUE | Word Sense Disambiguation |

Table 1: Datasets Description

prompts learned from previous dissimilar tasks can negatively influence the current task prompt. We wanted to see if our method, Selective Progressive Prompts, would address this issue. We compared our method using two similarity thresholds (0.7 and 0.85) with per-task prompt tuning (without forward transfer) and progressive prompts.

#### 4.3.1 Prompt Length

We used the same training parameters from Progressive Prompt's T5 experiments [4] when training soft prompts in our experiments. For both Progressive Prompt and Selective Progressive Prompt, we set the prompt token length to 50. This is because half of the prompt tokens are tunable in progressive prompt tuning and the other half are frozen prompts from previous tasks. For Prompt Tuning (without Progressive Networks and forward transfer), we set the prompt token length to 100. This is because prompt tuning does not concatenate frozen prompts from previous tasks and it trains a separate per-task prompt, and thus the entire prompt tokens are tunable.

#### 4.3.2 Similarity Thresholds

In our experiments, we chose similarity thresholds of 0.70 and 0.85. When calculating the similarity score between two inputs from the task, we would evaluate whether the computed score surpasses or falls below these predefined thresholds. If it doesn't meet the threshold, it will not be used in the Selective Progressive Prompts, and we will train a separate per-task prompt using Prompt Tuning [2]. Conversely, if it meets the threshold then it will be included in our Selective Progressive Prompts. We chose these two thresholds as we have seen throughout calculating similarity that the scores are within the range of 0.65 to 0.9. Opting for a higher similarity threshold ensures that our method selectively incorporates task prompts based on prior prompts from closely related tasks, albeit with the potential trade-off of disregarding insights from tasks deemed less similar.

## 5 Results

For all Forward Transfer experiments, we perform two runs of the same experiment and we take the average accuracy results. We compare Selective Progressive Prompts with other approaches in Table 2 and Figure 2. Our approach on three out of the six experiments (IMDb-amazon, Yelp Review full-qqp, and WiC-Yelp Review Full) has the highest accuracy out of all the approaches.

We achieved better results than progressive prompts on five out of the six experiments (IMDb-copa, IMDb-Amazon, Yelp review full-QQP, MRPC-QQP, and WiC-Yelp review full). We also see that Selective Progressive Prompts with a threshold of 0.85 for task pair SST2-MRPC yield a higher result than 0.7 thresholds. For five out of six task pairs that result in a higher accuracy score than Progressive Prompts, we see an average improvement of 0.8% with our method. The strongest improvement we have seen is with IMDB-Amazon with a 1.7% improvement increase from progressive prompt and a 2.2% improvement from Prompt Tuning. We also see a 19.48% improvement in WiC-Yelp Review full when compared to Prompt Tuning and a 0.48% improvement when compared to Progressive Prompts.

| Tasks | Prompt Tuning (w/o transfer) | Progressive Prompts | Selective Progressive Prompts with 0.7 threshold (ours) | Selective Progressive Prompts with 0.85 threshold (ours) |
|---|---|---|---|---|
| imdb, copa | **52.0** | 49.0 | 50.0 | 48.0 |
| imdb, amazon | 60.56 | 61.06 | **62.76** | 62.44 |
| sst2, mrpc | **87.3** | 86.3 | 85.0 | 86.03 |
| yelp review full, qqp | 90.2 | 89.4 | **90.4** | 87.6 |
| mrpc, qqp | **91.2** | 90.0 | 90.6 | 89.9 |
| wic, yelp review full | 45.32 | 64.32 | **64.8** | 64.56 |

Table 2: Transfer learning results on six pairs of tasks from different domains. All results are averaged over 2 runs. The results (highlighted in bold) have the highest accuracy out of all the approaches. We achieved better results than progressive prompts on five out of the six pairs of tasks and the best performance on three out of the six pairs.

## 6 Discussion

Based on the results in Table 2, our approach with a 0.7 threshold had an average improvement of 0.8% from the original Progressive Prompts, with the exception of one task pair (SST2 - MRPC). Concatenating and learning from only the task prompts that are sufficiently similar allows the model to learn from relevant tasks without irrelevant prompts hindering the accuracy of the model. From these results, it can be seen that forward transfer benefits from concatenating task prompts when the tasks are sufficiently similar. Therefore, it is imperative that a selection criterion exists to only select task prompts that are sufficiently relevant enough to be included in the concatenation of prompts during forward transfer learning. Our selective approach helps the model to select relevant task prompts when learning a given task, thereby improving the accuracy and performance of transfer learning when learning a sequence of tasks.

On the other hand, for SST2 and MRPC, Selective Progressive Prompts has lower accuracy scores of 85.0 percent and 86.03 percent for the similarity thresholds of 0.7 and 0.85, respectively, compared to the Prompt Tuning accuracy of 87.3 percent. One possible reason for this lower accuracy is that SST2 is a sentiment analysis task while MRPC is a paraphrase detection task, which are two different tasks from different domains that have a higher dissimilarity compared to other task pairs. This means that forward transfer will only be beneficial for pairs of tasks that are

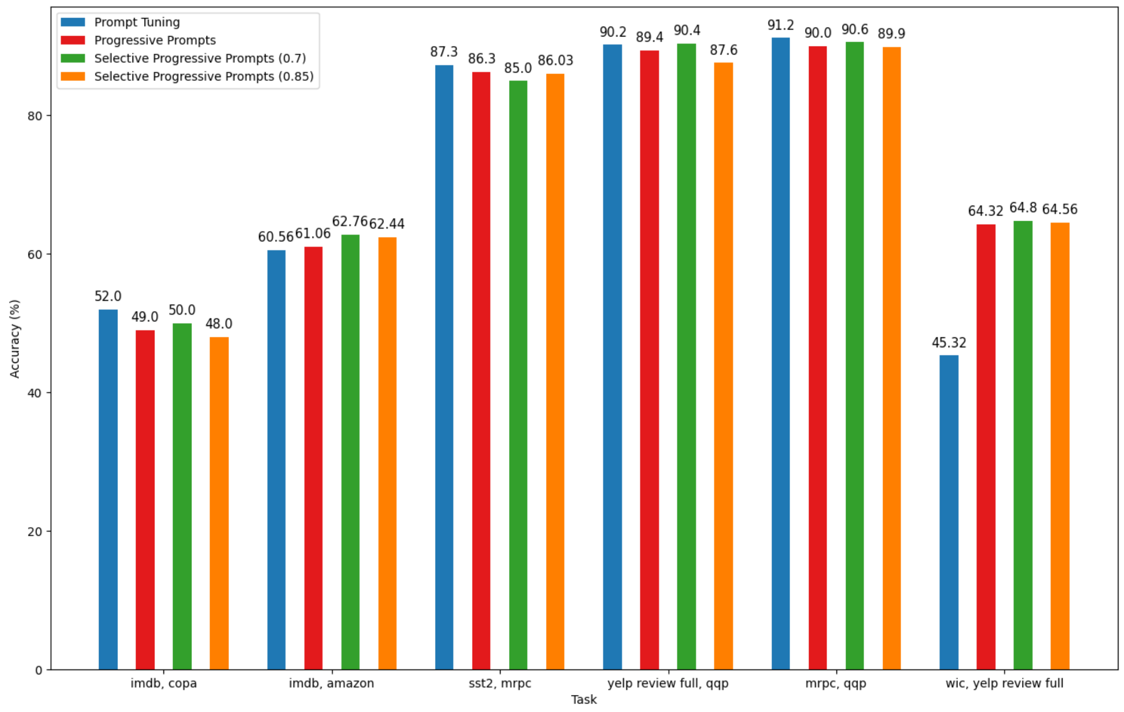


Figure 2: Our Forward Transfer experimental results comparing our method with per-task Prompt Tuning and Progressive Prompts

more similar to each other, and as a result, training a separate per-task prompt is better suited in cases where tasks are too different to reap the benefits of Progressive Prompts. This can be seen in other tasks from differing domains (IMDB - COPA) or differing datasets (MRPC - QQP) which benefit from training a separate per-task prompt using Prompt Tuning.

## 7 Conclusion

In this paper, we have introduced Selective Progressive Prompts, a modified version of Progressive Prompts that aims to improve forward transfer on a sequence of heterogeneous tasks in continual learning. Instead of concatenating every prompt from prior tasks like what Progressive Prompts does, we perform a cosine similarity check between each input embedding and the input embeddings from prior tasks. Based on the similarity check, we either train a separate per-task prompt using Prompt Tuning, or we use Progressive Prompts to concatenate the prompts from relevant tasks. Our selective approach allows us to learn prompts only using prompts from relevant tasks, thereby improving model accuracy when learning a task sequence with varying task similarities. For future work, we would like to test with different similarity thresholds, perhaps having an adaptive threshold based on the task and model performance. The similarity threshold parameter could be adaptively tuned instead of being fixed for different sets of tasks. We would also like to conduct more experiments and see the resulting performance with a larger sequence of tasks. With Selective Progressive Prompts, we aspire to contribute to the development of robust language model methods capable of integrating knowledge from a broad spectrum of tasks in continual learning. Our intention is to develop robust language models and methods that can adeptly adjust to a wide variety of different tasks, enabling dynamic, long-term language understanding in evolving real-world contexts.

## 8 Author Contributions

We jointly worked on the literature review, project ideation, implementation, and evaluation.

## References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*, 2019.

[2] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[3] Colin Raffel, Noam M. Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2019.

[4] Anastasia Razdaibiedina, Yuning Mao, Rui Hou, Madian Khabsa, Mike Lewis, and Amjad Almahairi. Progressive prompts: Continual learning for language models. In *International Conference on Learning Representations*, 2023.

[5] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *ArXiv*, abs/1606.04671, 2016.

[6] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *ArXiv*, abs/2302.13971, 2023.

[7] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. *SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems.* Curran Associates Inc., Red Hook, NY, USA, 2019.

[8] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In Tal Linzen, Grzegorz Chrupała, and Afra Alishahi, editors, *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[9] Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q Weinberger, and Yoav Artzi. Revisiting few-sample {bert} fine-tuning. In *International Conference on Learning Representations*, 2021.

[10] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.