MECHANISTICALLY ANALYZING THE EFFECTS OF FINE-TUNING ON PROCEDURALLY DEFINED TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Fine-tuning large pre-trained models has become the *de facto* strategy for developing models that are safe to deploy. However, there has been little work that explains how fine-tuning alters the underlying capabilities learnt by a model during pretraining: does fine-tuning yield entirely novel capabilities or does it just modulate existing ones? We address this question empirically in *synthetic* settings with mechanistic interpretability tools (e.g., network pruning and probing) to understand how the model's underlying capabilities are changing. Our extensive analysis of the effects of fine-tuning shows: (i) fine-tuning rarely alters the underlying model capabilities; (ii) a minimal transformation, which we call a 'wrapper', is typically learned on top of the underlying model capabilities; and (iii) further fine-tuning on a task where such wrapped capabilities are relevant leads to sample-efficient "revival" of the capability, i.e., the model begins reusing this capability in a few gradient steps. *This indicates practitioners can unintentionally remove a model's safety wrapper by merely fine-tuning it on a superficially unrelated task.*

1 INTRODUCTION

Large language models (LLMs) pretrained on huge, web-crawled text datasets demonstrate extremely general capabilities [1–4]. This has led to the current paradigm of machine learning, where practitioners often use model adaptation protocols such as fine-tuning to achieve unprecedented performance on a broad range of downstream tasks [5–9]. Fine-tuning with different training objectives has also seen immense usage in mitigating "unsafe" capabilities (e.g., producing sensitive, biased, or toxic outputs), serving as an integral component of current state-of-the-art *alignment* approaches like RLHF [10–14]. Given the ubiquity of fine-tuning, a natural question emerges: precisely how does fine-tuning influence a pretrained model's capabilities to adapt it to the downstream dataset? The generality of an LLM's capabilities makes it feasible that fine-tuning protocols merely identify the most relevant capability and amplify its use for a given set of inputs, while inhibiting the use of other capabilities that are viable for processing the inputs. This can be concerning in scenarios where fine-tuning is used for removing undesirable behaviors from a model, as the protocol may just "hide" the undesirable capability.

Motivated by the above, we perform an extensive analysis of the effects of fine-tuning on a pretrained model's capabilities in *controlled settings* where we can use mechanistic interpretability tools to understand precisely what is happening to the model's underlying capbilities. Specifically, we focus on compiled transformer models based on the *Tracr library* [15, 16]—which allows encoding specific computational programs into a transformer—and procedurally generated setups involving *probabilistic context-free grammars (PCFGs)* [17, 18]—a formal model designed to capture syntactic properties of natural and programmatic languages that has recently served as a testbed for mechanistically understanding language models [19–21]. While Tracr allows us to analyze models with perfectly encoded capabilities, models trained on PCFGs allow us to evaluate the effects of different pretraining design choices that may yield learning of "approximate" capabilities. Fine-tuning these models via the often used protocol of further training a pretrained model on a downstream dataset with a sufficiently small learning rate, we find when a "relevant" pretraining capability is present, the fine-tuned model learns a minimally transformed version of it. We call this transformation a wrapper and find that the wrapper can often be extremely localized, e.g., via mere pruning of a few weights or neurons, the model can start to reuse its pretraining capability and forget how to perform the downstream task. Morover, further fine-tuning the model on a subset of pretraining data leads to an extremely sample-efficient revival of the capability.

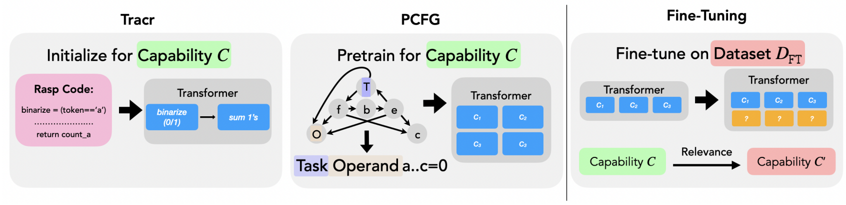


Figure 2: **Experimental setup.** We primarily analyze two setups: (i) Tracr "compiled" models with predefined capabilities and (ii) models trained to learn capabilities defined via a PCFG [19]. During fine-tuning, we train the model on a dataset $\mathcal{D}_{\mathtt{FT}}$ that promotes learning of a capability $\mathcal{C}'$. We randomly embed spurious features in the dataset such that features extracted by a pretraining capability $\mathcal{C}$ may exploit the feature to reduce loss on the dataset.

2 Defining our notion of capabilities

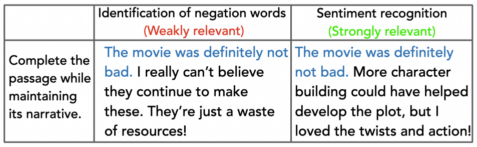
For precision, we first discuss the notion of capabilities that we aim to capture for analyzing how fine-tuning alters a model. Let $\mathcal{D}_{\mathtt{PT}}$ denote a dataset sampled from a distribution $\mathcal{P}_{\mathrm{X}}$ over the domain X. We will assume the domain X can itself be factorized into two domains X_I and X_D. Correspondingly, a sample $x \in \mathrm{X}$ can be divided into a tuple of variables $(x_i \in \mathrm{X}_I, x_d \in \mathrm{X}_D)$, where x_i identifies which capability a model should use to process the information encoded by the variable x_d. This decomposition captures the idea that different prompts can force a pretrained LLM to elicit different capabilities, e.g., by adding the phrase "think step by step" [22]. The identifier of capability c is denoted i_c. Pretraining on $\mathcal{D}_{\mathtt{PT}}$ yields us a model $\mathtt{M}(.)\colon \mathrm{X} \rightarrow \mathrm{Y}$, where we distinguish between the domain and co-domain of the model for the purpose of our framework; in practice, one can assume $\mathrm{Y} = \mathrm{X}$ for language models trained on a sufficiently large dataset. We define a capability as follows.

Definition 1. (Capability.) *Define a map $\mathcal{C} : \mathrm{X}_{\mathcal{D}} \rightarrow \mathrm{Y}_{\mathcal{C}}$, where $\mathrm{Y}_{\mathcal{C}} \subset \mathrm{Y}$. Let $\mathrm{X}_{\mathcal{C}} \subset \mathrm{X}$ be a sub-domain such that for all $x \in X_{\mathcal{C}}$, the capability identifier variable is the same, i.e., $x_i = \mathtt{i}_{\mathcal{C}}$. Then, we say the model $\mathtt{M}$ "possesses a capability $\mathcal{C}$" if for all $x \in \mathrm{X}_{\mathcal{C}}$, $\mathtt{M}(x) = \mathcal{C}(x_d)$.*

We use an idealized definition to relay our primary intuition and emphasize that we do not expect all capabilities a pretrained model seems to possess will act as perfectly as the definition necessitates. We next consider how the fine-tuning distribution $\mathcal{P}_X^{\mathtt{FT}}$ over the domain X can interact with capabilities exhibited in the pretrained model. Our goal here is to capture the fact that a large-scale pretraining corpora is likely to have non-zero probability under the fine-tuning distribution. Correspondingly, it is unlikely that a pretrained model will lack *any* capability relevant to the downstream task. This motivates a notion of "relevance of a capability". Specifically, let $\mathcal{D}_{\mathtt{FT}} \sim \mathcal{P}_X^{\mathtt{FT,E}}$ denote the downstream dataset used for fine-tuning, where $\mathcal{P}_X^{\mathtt{FT,E}}$ is the empirical distribution that captures a subset of the support with non-zero probability in the distribution $\mathcal{P}_X^{\mathtt{FT}}$.

Definition 2. (Relevance of a Capability.) *Assume the capability $\mathcal{C}$ possessed by the pretrained model $\mathtt{M}(.)$ can be transformed to a map $\mathtt{g} \circ \mathcal{C}$ via fine-tuning on $\mathcal{D}_{\mathtt{FT}}$, such that for all $x \sim \mathcal{P}_X^{\mathtt{FT,E}}$, the correct output is produced. Then, if for all $x \sim \mathcal{P}_X^{\mathtt{FT}}$, $\mathtt{g} \circ \mathcal{C}$ yields the correct output, we claim capability $\mathcal{C}$ is **strongly relevant** to the fine-tuning task; else, we call it **weakly relevant**.*

For example (also see Fig. 1), a *weakly relevant* capability can involve the ability to recognize a spurious feature that the model can learn to exploit to perform well on the fine-tuning dataset, without enabling generalization to the overall distribution that the fine-tuning dataset is sampled from. Meanwhile, a *strongly relevant* capability is one that extract a causally relevant feature for that task. When a weakly relevant capability is used by the model, we find we can often localize neurons which implement the transform $\mathtt{g}$ in Def. 2. We call such a $\mathtt{g}$ a **wrapper** and $\mathtt{g} \circ \mathcal{C}$ a **wrapped capability**.



	Identification of negation words (Weakly relevant)	Sentiment recognition (Strongly relevant)
Complete the passage while maintaining its narrative.	The movie was definitely not bad. I really can't believe they continue to make these. They're just a waste of resources!	The movie was definitely not bad. More character building could have helped develop the plot, but I loved the twists and action!

Figure 1: **Capability Relevance.** Consider the task of completing a passage while maintaining its narrative. Herein, the ability to recognize the sentiment of a text will be deemed *strongly relevant* and the ability to recognize negative words *weakly relevant*. Such words are often correlated with a negative sentiment.

3 Building Capable Models: Tracr and PCFGs

Due to the lack of clarity on what capabilities a language model possesses or what training data it has seen, we primarily focus on controllable, procedurally generated setups for our analysis (see Fig. 2).

To model relevance of pretraining capabilities, we randomly embed into the fine-tuning dataset a feature that spuriously correlates with some feature extracted via a pretraining capability.

Compiled capabilities with Tracr. For a fully controllable system with no ambiguities in model capabilities, we use Tracr [15], a recently proposed library that enables "compiling" a transformer model with a set of predefined computational primitives over a string of characters from the English alphabet, such as counting and sorting.

- **Counter (C):** Compile the capability to count the number of occurrences of a token $\mathtt{t_{pre}}$ into the model; fine-tune to count occurrences of another token $\mathtt{t_{FT}}$. If $\mathtt{n}(x, t)$ denotes the number of occurrences of a token t in a string x, the spurious correlation is defined by enforcing a constant difference in token occurrences, i.e., $\mathtt{n}(x, \mathtt{t_{FT}}) - \mathtt{n}(x, \mathtt{t_{pre}}) = \mathtt{q}$.

We also consider a "Max-identifier" task, which is deferred to the appendix due to space constraints.

Learned capabilities with PCFGs. In this setup, the capabilities are "learned", akin to practical situations. This allows us to probe the effects of different pretraining design decision, e.g., the number of pretraining iterations or the distribution of data. Specifically, we follow recent work [19] and train a minGPT model [23] via autoregressive modeling on probabilisitc context-free grammars (PCFGs), a formal model of language that captures syntactic properties. We prepend special tokens $\mathtt{T}$ and $\mathtt{O}$, called task and operand tokens, that specify a certain task must be performed on the string x based upon the operand $\mathtt{O}$; e.g., count the occurrences (task) of a certain token (operand) in the string. The output of the task and a special token indicating that the output should be produced at the next position are appended at the end of the string (see Appendix D.2 for further details). During sampling, we use a uniform prior over task tokens and a multinomial prior over the set of operand tokens, denoted $\{\mathtt{O}\}$; probability of sampling token $\mathtt{O} \in \{\mathtt{O}\}$ under task $\mathtt{T}$ is denoted $\mathcal{P}_{\mathtt{T}}(\mathtt{O})$. If this probability is low, the model may not learn the relevant capability to perform the task on operand $\mathtt{O}$. Again, we analyze two tasks, but discuss only the following in the paper due to space constraints.

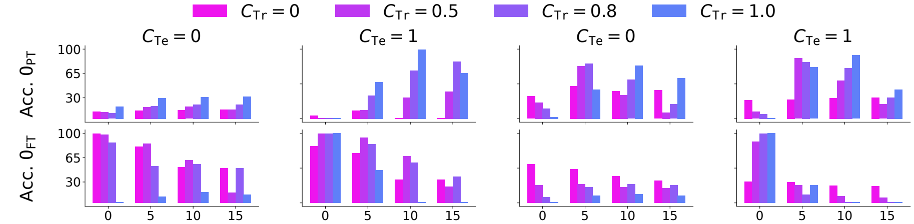
- **Counter (C):** We intentionally reuse this task to understand the effects of compilation of learning a capability, instead of perfectly compiling it. The setup is similar to Tracr's, but the data is now sampled from a PCFG and the model is *trained* to, amongst five other tasks (see appendix for details), count the occurrences of a *set* of tokens $\{\mathtt{O_{pre}}\}$. This model is then fine-tuned to count occurrences of a specific token $\mathtt{O_{FT}}$. By altering this token's sampling probability during pretraining, we can assess the impact of the distribution used for pretraining on fine-tuning.

4 Results and Discussion

In this section, we provide several results indicating that fine-tuning rarely elicits a meaningful change of pretraining capabilities. We heavily utilize **network pruning** [24, 25], **attention map visualizations** [26–28], and **probing classifiers** [29–32] as tools for mechanistic interpretability analysis. Additionally, we propose a methodology called **reverse fine-tuning** (`reFT`), wherein one takes a pretrained model ($\mathtt{M}$), fine-tunes it on a downstream dataset ($\mathtt{M_{FT}}$), and then fine-tunes it again in the "reverse" direction, i.e., on a dataset sampled from the original pretraining distribution ($\mathtt{M_{reFT}}$). We argue that if the behavior corresponding to a capability from $\mathtt{M}$ is retrieved in a few steps of `reFT` , fine-tuning did not meaningfully alter said capability. We note that we primarily focus on the learned capabilities setup of PCFG Counters in this section, relegating most results on compiled capabilities to App. I, learned capabilities to App. J, and also some preliminary analysis of the recently released TinyStories dataset [33] to App. H—results are consistent across all settings. In the PCFG Counters setup, the model is pretrained to count tokens $\{\mathtt{O_{pre}}\} = \{a, b, c\}$ in a given string; during fine-tuning, the model is trained to count the token $\mathtt{O_{FT}} = \mathtt{b}$, wherein the spurious correlation is defined by enforcing count of $\mathtt{b}$ to be 1 larger than that of $\mathtt{a}$. The probability of embedding a spurious correlation in the train/test fine-tuning dataset is denoted $R_{\mathtt{FT,Tr}}/R_{\mathtt{FT,Te}}$, with $R_{\mathtt{FT,Tr}} \in \{0.0, 0.5, 0.8, 1.0\}$ and $R_{\mathtt{FT,Te}} \in \{0.0, 1.0\}$. We use three sets of sampling probabilities of the task operands in the pretraining data: $\mathcal{P}_{C}^{L} = (0.999, 0.001, 0.000)$, $\mathcal{P}_{C}^{M}(0.9, 0.1, 0.0)$, or $\mathcal{P}_{C}^{H} = (0.5, 0.3, 0.2)$. We also sweep the fine-tuning learning rate η, $\eta_L = 10^{-3}, \eta_M = 10^{-4}, \eta_S = 10^{-5}$.

Experiment 1: We first evaluate the model's learning dynamics during fine-tuning (see Fig. 6). When the pretraining prior has low probability of sampling the target token, the fine-tuned model performs well only when the spurious correlation is present, i.e., $R_{\mathtt{FT,Te}} = 1$. As the prior is reduced, however, we observe this behavior significantly changes. Only having a sufficiently large sampling probability of the token during pretraining leads the model to avoid the spurious correlation. We argue this is likely to happen due to the model partially learning the strongly relevant capability, with fine-tuning enabling the completion of this learning. Based on this behavioral analysis, we hypothesize that the model learns a wrapper on the *weakly relevant capability* of counting the token $\texttt{O}_{\texttt{pre}}$, adding the relevant difference to yield the correct result for $\texttt{O}_{\texttt{FT}}$. To evaluate this, we analyze the models fine-tuned with a low sampling prior via network pruning and linear probing (see Fig. 8). The details of pruning are discussed in the appendix. If the model learns a wrapper on this capability, deleting the wrapper neurons should recover the capability to count $\texttt{O}_{\texttt{pre}}$. As shown in Fig. 3, we find this is indeed the case. To confirm this hypothesis further, we perform probing analysis by training a linear layer on every block's residual output. As shown in Fig. 8 (a), in the presence of spurious correlations, the model with weakly relevant capability is able to perform well on counting the token $\texttt{O}_{\texttt{pre}}$ when fine-tuned using correlated data.

Takeaway: We find the model seems to exploit the spurious correlation when the sampling prior of target token is low during pretraining, but avoids it when the prior is sufficiently high to enable partial learning of the strongly relevant capability. Mechanistic analysis shows that the features learned by the weakly relevant capability can be localized in the model, indicating learning of a wrapper.

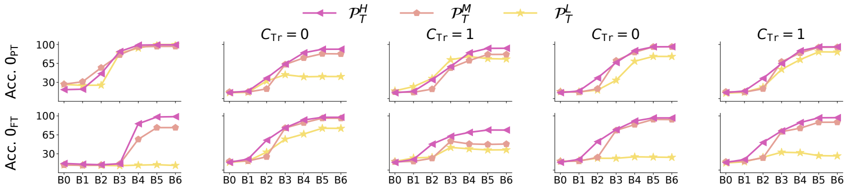


(a) η_M (b) η_S

Figure 3: **Pruning analysis: Fine-tuning task related features learned by the weakly relevant capability can be localized in the model:** Evaluation is done for the pre-training task of counting the token $\texttt{O}_{\texttt{pre}}$. The pre-training task capability can be revived on using a low learning rate η_S.

Experiment 2: The above analysis focuses on the learning of a capability relevant to the downstream task. We note analyze how the pre-training capability itself is affected during fine-tuning. Specifically, we use a model fine-tuned to count $\texttt{O}_{\texttt{FT}}$ and reverse fine-tune (`reFT`) it to learn its original pre-training capability to count $\texttt{O}_{\texttt{pre}}$. As shown in Fig. 9 (a), even with a small learning rate (η_S), the pretraining capability is revived via `reFT` ; this revival is made quicker with increase in $\mathcal{P}_{\texttt{C}}(\texttt{O}_{\texttt{FT}})$, as shown in Fig. 9 (appendix). This *indicates* that upon fine-tuning, relevant "circuits" for the pre-training capability continue to persist in the model. To analyze this further, we train linear probes on intermediate representations from each block of the model to produce the output of the pre-training task of counting $\texttt{O}_{\texttt{pre}}$. If the pre-training capabilities are indeed present in the model after fine-tuning to count on token $\texttt{O}_{\texttt{FT}}$, the probes should yield us high performance. As shown in Fig. 8, the pre-training capabilities to count $\texttt{O}_{\texttt{pre}}$ is indeed present in the model, especially under strong-relevance. Under weak-relevance of the capability, a smaller fine-tuning learning rate (η_S), as expected to be used in a practical scenario, leads to continued existence of the pre-training capability.

Takeaway: We find that the pre-training capability is not lost on fine-tuning.



(a) Pre-trained Model (b) η_M (c) η_S

Figure 4: **Probing the presence of pre-training (top) and fine-tuning (bottom) capabilities.** We plot probe accuracy versus the index of the block in the Transformer model. $C_{\texttt{Te}}$ is set to 0. The pretrained model (left) acts as a baseline for the trend of performance through the model's blocks. We find we can infer the count of $\texttt{O}_{\texttt{PT}}$ with a similar trend as the pretrained model (left). A drop in performance is observed only when learning rate η_M is used with a weakly relevant capability (low sampling prior). This indicates pretraining capabilities continues to persist upon fine-tuning.

References

[1] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[2] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[4] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[5] Colin Raffel, Noam 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. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[6] Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M Rush. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=9Vrb9D0WI4.

[7] Machel Reid, Yutaro Yamada, and Shixiang Shane Gu. Can wikipedia help offline reinforcement learning? *arXiv preprint arXiv:2201.12122*, 2022.

[8] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

[9] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[10] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[11] Dongyoung Go, Tomasz Korbak, Germán Kruszewski, Jos Rozen, Nahyeon Ryu, and Marc Dymetman. Aligning language models with preferences through f-divergence minimization. *arXiv preprint arXiv:2302.08215*, 2023.

[12] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021, 2020.

[13] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

[14] Amelia Glaese, Nat McAleese, Maja Trębacz, John Aslanides, Vlad Firoiu, Timo Ewalds, Maribeth Rauh, Laura Weidinger, Martin Chadwick, Phoebe Thacker, et al. Improving alignment of dialogue agents via targeted human judgements. *arXiv preprint arXiv:2209.14375*, 2022.

[15] David Lindner, János Kramár, Matthew Rahtz, Thomas McGrath, and Vladimir Mikulik. Tracr: Compiled transformers as a laboratory for interpretability. *arXiv preprint arXiv:2301.05062*, 2023.

[16] Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like transformers. In *International Conference on Machine Learning*, pages 11080–11090. PMLR, 2021.

[17] Michael Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29, 1996.

[18] Noam Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

[19] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 1, context-free grammar. *arXiv preprint arXiv:2305.13673*, 2023.

[20] Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, et al. Neural networks and the chomsky hierarchy. *arXiv preprint arXiv:2207.02098*, 2022.

[21] Hui Shi, Sicun Gao, Yuandong Tian, Xinyun Chen, and Jishen Zhao. Learning bounded context-free-grammar via lstm and the transformer: Difference and the explanations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 8267–8276, 2022.

[22] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.

[23] Andrej Karpathy. *MinGPT*, 2020. Github link. https://github.com/karpathy/minGPT/tree/master.

[24] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*, 2019.

[25] Hidenori Tanaka, Aran Nayebi, Niru Maheswaranathan, Lane McIntosh, Stephen Baccus, and Surya Ganguli. From deep learning to mechanistic understanding in neuroscience: the structure of retinal prediction. *Adv. in Neural Information Processing Systems (NeurIPS)*, 2019.

[26] Sofia Serrano and Noah A Smith. Is attention interpretable? *arXiv preprint arXiv:1906.03731*, 2019.

[27] Sarah Wiegreffe and Yuval Pinter. Attention is not not explanation. *arXiv preprint arXiv:1908.04626*, 2019.

[28] Vivian Lai and Chenhao Tan. On human predictions with explanations and predictions of machine learning models: A case study on deception detection. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 29–38, 2019.

[29] Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*, 2019.

[30] Elena Voita and Ivan Titov. Information-theoretic probing with minimum description length. *arXiv preprint arXiv:2003.12298*, 2020.

[31] Mor Geva, Jasmijn Bastings, Katja Filippova, and Amir Globerson. Dissecting recall of factual associations in auto-regressive language models. *arXiv preprint arXiv:2304.14767*, 2023.

[32] Mor Geva, Avi Caciularu, Kevin Ro Wang, and Yoav Goldberg. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. *arXiv preprint arXiv:2203.14680*, 2022.

[33] Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.

[34] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[35] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021.

[36] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

[37] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965, 2022.

[38] Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, et al. A general language assistant as a laboratory for alignment. *arXiv preprint arXiv:2112.00861*, 2021.

[39] 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 *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[40] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*, 2021.

[41] Yihan Wang, Si Si, Daliang Li, Michal Lukasik, Felix Yu, Cho-Jui Hsieh, Inderjit S Dhillon, and Sanjiv Kumar. Two-stage llm fine-tuning with less specialization and more generalization. *arXiv preprint arXiv:2211.00635*, 2022.

[42] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[43] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. Adapterhub: A framework for adapting transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020): Systems Demonstrations*, pages 46–54. Association for Computational Linguistics, 2020.

[44] Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.15647*, 2023.

[45] Almog Gueta, Elad Venezian, Colin Raffel, Noam Slonim, Yoav Katz, and Leshem Choshen. Knowledge is a region in weight space for fine-tuned language models. *arXiv preprint arXiv:2302.04863*, 2023.

[46] Michael S Matena and Colin A Raffel. Merging models with fisher-weighted averaging. *Advances in Neural Information Processing Systems*, 35:17703–17716, 2022.

[47] Andrew K Lampinen and Surya Ganguli. An analytic theory of generalization dynamics and transfer learning in deep linear networks. *arXiv preprint arXiv:1809.10374*, 2018.

[48] Nilesh Tripuraneni, Michael Jordan, and Chi Jin. On the theory of transfer learning: The importance of task diversity. *Advances in neural information processing systems*, 33:7852–7862, 2020.

[49] Federica Gerace, Luca Saglietti, Stefano Sarao Mannelli, Andrew Saxe, and Lenka Zdeborová. Probing transfer learning with a model of synthetic correlated datasets. *Machine Learning: Science and Technology*, 3(1):015030, 2022.

[50] Wesley Maddox, Shuai Tang, Pablo Moreno, Andrew Gordon Wilson, and Andreas Damianou. Fast adaptation with linearized neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 2737–2745. PMLR, 2021.

[51] Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. Fine-tuning can distort pretrained features and underperform out-of-distribution. *arXiv preprint. arXiv:2202.10054*, 2022.

[52] Greg Yang and Edward J Hu. Feature learning in infinite-width neural networks. *arXiv preprint arXiv:2011.14522*, 2020.

[53] Sadhika Malladi, Alexander Wettig, Dingli Yu, Danqi Chen, and Sanjeev Arora. A kernel-based view of language model fine-tuning. In *International Conference on Machine Learning*, pages 23610–23641. PMLR, 2023.

[54] Jeevesh Juneja, Rachit Bansal, Kyunghyun Cho, João Sedoc, and Naomi Saphra. Linear connectivity reveals generalization strategies. *arXiv preprint. arXiv:2205.12411*, 2022.

[55] Ekdeep Singh Lubana, Eric J. Bigelow, Robert P. Dick, David Krueger, and Hidenori Tanaka. Mechanistic Mode Connectivity, 2022. Comment: 39 pages.

[56] Charles Lovering, Rohan Jha, Tal Linzen, and Ellie Pavlick. Predicting inductive biases of pre-trained models. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=mNtmhaDkAr.

[57] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*, 2020.

[58] Suhas Kotha, Jacob Mitchell Springer, and Aditi Raghunathan. Understanding catastrophic forgetting in language models via implicit inference. *arXiv preprint arXiv:2309.10105*, 2023.

[59] Haoyu Zhao, Abhishek Panigrahi, Rong Ge, and Sanjeev Arora. Do transformers parse while predicting the masked word? *arXiv preprint arXiv:2303.08117*, 2023.

[60] Kenneth Li, Aspen K. Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent World Representations: Exploring a Sequence Model Trained on a Synthetic Task, 2023. Comment: ICLR 2023 oral (notable-top-5%): https://openreview.net/forum?id=DeG07_TcZvT ; code: https://github.com/likenneth/othello_world.

[61] Neel Nanda, Andrew Lee, and Martin Wattenberg. Emergent linear representations in world models of self-supervised sequence models. *arXiv preprint arXiv:2309.00941*, 2023.

[62] Bingbin Liu, Jordan T Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. *arXiv preprint arXiv:2210.10749*, 2022.

[63] Josef Valvoda, Naomi Saphra, Jonathan Rawski, Adina Williams, and Ryan Cotterell. Benchmarking compositionality with formal languages. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 6007–6018, 2022.

[64] Bingbin Liu, Jordan T Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Exposing attention glitches with flip-flop language modeling. *arXiv preprint arXiv:2306.00946*, 2023.

[65] Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Josh Susskind, Samy Bengio, and Preetum Nakkiran. What algorithms can transformers learn? a study in length generalization. *arXiv preprint arXiv:2310.16028*, 2023.

[66] Stephanie Chan, Adam Santoro, Andrew Lampinen, Jane Wang, Aaditya Singh, Pierre Richemond, James McClelland, and Felix Hill. Data distributional properties drive emergent in-context learning in transformers. *Advances in Neural Information Processing Systems*, 35:18878–18891, 2022.

[67] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.1, knowledge storage and extraction. *arXiv preprint arXiv:2309.14316*, 2023.

[68] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.2, knowledge manipulation. *arXiv preprint arXiv:2309.14402*, 2023.

[69] Maya Okawa, Ekdeep Singh Lubana, Robert P Dick, and Hidenori Tanaka. Compositional abilities emerge multiplicatively: Exploring diffusion models on a synthetic task. *https://openreview.net/forum?id=ZXH8KUgFx3*, 2023.

[70] Xiangyu Qi, Yi Zeng, Tinghao Xie, Pin-Yu Chen, Ruoxi Jia, Prateek Mittal, and Peter Henderson. Fine-tuning aligned language models compromises safety, even when users do not intend to! *arXiv preprint arXiv:2310.03693*, 2023.

[71] Xianjun Yang, Xiao Wang, Qi Zhang, Linda Petzold, William Yang Wang, Xun Zhao, and Dahua Lin. Shadow alignment: The ease of subverting safely-aligned language models. *arXiv preprint arXiv:2310.02949*, 2023.

[72] Anonymous. Dissecting learning and forgetting in language model finetuning. In *Submitted to The Twelfth International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=tmsqb6WpLz. under review.

[73] Anonymous. Learning and forgetting unsafe examples in large language models. In *Submitted to The Twelfth International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=hkQOYyUChL. under review.

[74] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[75] Ekdeep Singh Lubana and Robert P. Dick. A gradient flow framework for analyzing network pruning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=rumv7QmLUue.

[76] Michael C Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. *Advances in neural information processing systems*, 1, 1988.

[77] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does llm safety training fail? *arXiv preprint arXiv:2307.02483*, 2023.

[78] Andy Zou, Zifan Wang, J Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.

[79] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

A Organization of Appendix

In the appendix we present a comprehensive analysis of our claims on Tracr, PCFG and TinyStories-Instruct using different mechanistic interpretability tools discussed in Section-F of the main paper. We also present a summary of the notations used in this work in Tab. 1. Overall, the appendix is organized as follows:

- Sec. C presents main results on Tracr and Tin0-stories to support our hypothesis. We also present addition results for PCFG in this section.
- Sec. D presents details of the Tracr, PCFG and Tiny Stories datasets utilized in this work.
- Sec. E presents the training and model details for each of the datasets considered.
- Sec. F lists the protocols used for different mechanistic interpretability tools like attention maps, probing, pruning and reverse fine-tuning.
- Sec. G provides a few more results in practically relevant contexts, such as in a synthetic jailbreaking setup.
  - Sec. G.1 studies the effect of using different fractions of pre-training and fine-tuning data points for fine-tuning.
  - Sec. G.2 presents the jailbreaking analysis using the PCFG setup.
  - Sec. G.3 shows reverse fine-tuning a fine-tuned model is sample efficient compared to baselines for both PCFG and Tracr models.
  - Sec. G.4 presents reverse fine-tuning analysis of a fine-tuning protocol that actively tries to remove a capability from PCFG / Tracr models.
- Sec. H presents detailed discussion of setup details and results on TinyStories.
- Sec. I presents additional results on Tracr for counter and max element tasks.
- Sec. J presents additional results on PCFG for the counting and index of occurrence tasks.

B Related Work

Fine-tuning in the "foundation model" era. Fine-tuning large-scale foundation models pretrained on huge datasets, such as LLMs [2, 3] or large vision models [34, 35], has become the norm in most domains of machine learning. Accordingly, several fine-tuning methods have been proposed in recent years, e.g., instruction fine-tuning [36–38], parameter-efficient fine-tuning [39–41], low-rank adaptation [42–44], and weight averaging [45, 46]. The diversity of these protocols makes fine-tuning a general, umbrella term for related methods used to adapt a pretrained model to elicit its most relevant capabilities. *For precision, we restrict this paper to fine-tuning protocols that continue training of a pretrained model on a smaller downstream dataset at a learning rate that is often one to three orders of magnitude lower than the average pretraining one.* Such protocols are widely used in practice, e.g., in instruction fine-tuning [36].

Understanding fine-tuning. A few papers theoretically analyze fine-tuning [47–51] under strong assumptions such as relatively simple model classes (e.g., linear functions) or a kernel view of deep learning, which, as shown by Yang and Hu [52], trivializes the notion of feature transfer in fine-tuning / transfer learning (though see Malladi et al. [53] for a notable exception). Prior works have also evaluated the effects of fine-tuning via the lens of mode connectivity [54, 55], behavioral evaluations [56], and intrinsic dimensionality of the loss landscape [57]. In contrast, we aim to provide a mechanistic analysis of how fine-tuning changes model capabilities. A relevant recent work by Kotha et al. [58] claims that fine-tuning is unlikely to alter a model's capabilities and supports this claim by providing a behavioral analysis of linear regression tasks and realistic models via a novel prompting strategy.

Model interpretability via synthetic tasks. Several recent works have focused on *mechanistically* understanding how Transformers learn synthetic language generation tasks, such as learning formal grammars and board games [19, 59–66]. The goal of such papers, including ours, is not necessarily to provide accurate explanations for the success of LLMs, but to develop concrete hypotheses that

Table 1: Notations used in this work.

Notation	Meaning
X	Input domain
X_D	Factor of the input domain that captures values of the inputs
X_I	Factor of the input domain that captures task identifiers
$\mathcal{P}_{\mathrm{X}}$	Probability distribution over the input domain
$\mathcal{P}_{\mathrm{X}}^{\mathtt{FT}}$	The overall distribution defining the downstream fine-tuning task
$\mathcal{D}_{\mathtt{FT}}$	Dataset used for fine-tuning
$\mathcal{P}_{\mathrm{X}}^{\mathtt{FT,E}}$	Empirical distribution from which the fine-tuning dataset is sampled
$\mathtt{T}$	Denotes a task to be performed by the model (e.g., count)
$\mathtt{O}$	Denotes an operand that will be processed by to perform the task $\mathtt{T}$
$\{\mathtt{O}_{\mathtt{PT}}\}$	Set of operand tokens seen during pretraining
$\mathtt{O}_{\mathtt{PT}}$	A specific token used as an operand during pretraining
$\mathtt{O}_{\mathtt{FT}}$	A specific token used as an operand during fine-tuning
$\mathtt{r}(x, \mathtt{O})$	Denotes the result of executing a task from Sec. 3 on a string x for some operand $\mathtt{O}$
$C_{\mathtt{Tr}}$	Probability that a randomly sampled string in the training data used for fine-tuning has a spurious correlation between the pretraining capability and the downstream task
$C_{\mathtt{Te}}$	Probability that a randomly sampled string in the test data used for evaluating fine-tuned models has a spurious correlation between the pretraining capability and the downstream task
$\mathcal{P}_{\mathtt{T}}(\mathtt{O})$	Sampling prior. Denotes the probability that when a string with task token $\mathtt{T}$ is sampled during pretraining, the operand to perform the task on is $\mathtt{O}$
$\mathcal{P}_C^H, \mathcal{P}_C^M, \mathcal{P}_C^S$	Sampling priors such that the probability of sampling the target token for fine-tuning ($\mathtt{O}_{\mathtt{FT}}$) is high ($\mathcal{P}_C^H$), medium ($\mathcal{P}_C^M$), or small ($\mathcal{P}_C^S$)
$\eta_M, \eta_S, \eta_{VS}$	Medium / Small / Very-small learning rates used for fine-tuning. η_{VS} is only used for a specific reverse fine-tuning experiment with Tracr compiled models.
$\mathtt{reFT}$	Denotes reverse fine-tuning
$n_{\mathtt{iters}}$	Number of iterations used during pre-training
LR	Learning rate

can be used to develop grounded experiments or tools for understanding their behavior. For example, in a recent work, Allen-Zhu and Li [67, 68] use a synthetically designed setup to develop hypotheses for how “knowledge” about an entity is stored in a pretrained model, showing such knowledge can often be manipulated via relatively simple linear transformations. Similarly, Okawa et al. [69] use a procedurally defined multimodal dataset to demonstrate that emergent capabilities seen in neural networks are partially driven by the compositional nature of real world data. In another work, Zhou et al. [65] utilize Tracr compiled Transformers to hypothesize and demonstrate that if primitive operations involved in a formal algorithm can be implemented by a model, length generalization if practically feasible.

C Additional Results

C.1 Validating our Hypotheses on Tracr

Attention map visualizations further corroborate the wrappers hypothesis. As we noted before, the results discussed above remain consistent across other experimental setups for both Tracr and PCFG models. However, by construction, Tracr yields particularly interpretable attention maps, allowing us to directly visualize the effects of fine-tuning. We thus analyze the attention maps of a Tracr model on the Counter task described in Sec. 3. Results are shown in Fig. 5. The original Tracr compiled model serves as a baseline and clearly demonstrates that all tokens only attend the pretraining target token, $\mathtt{O}_{\mathtt{PT}} = a$. Upon fine-tuning to count $\mathtt{O}_{\mathtt{FT}} = b$, we find the model clearly continues to pay attention to $\mathtt{O}_{\mathtt{PT}}$ if a small learning rate is used. A larger learning rate is, however, able to alter the model computation, but only if the pretraining capability is not weakly relevant to the fine-tuning task, i.e., when $C_{\mathtt{Tr}} = 0$; otherwise, we again find the model continues to pay attention to the pretraining target.

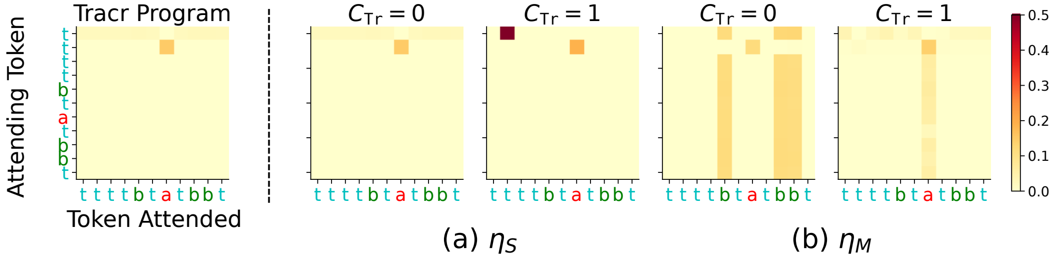


Figure 5: **Visualizing attention maps of fine-tuned Tracr models.** Leftmost panel shows the Tracr compiled model's attention map on the counter task. Upon fine-tuning under different spurious correlations, we see the model continues to pay attention to the pretraining target $\mathtt{O}_{\mathtt{PT}} = a$. Only when a large enough learning rate and zero spurious correlation is used, is there a change in the attention pattern.

C.2 Additional results validating our Hypotheses on PCFG

On probing, we observe that it is indeed possible to unwrap the wrapper learned by the models during fine-tuning. The results are shown in Fig-8. These observations therefore indicate that during fine-tuning the model can learn a minimal transform on its pre-training capabilities.

C.3 Validating our Hypotheses on TinyStories

To give additional strength to our results, we perform an analysis using more realistic language models trained on the TinyStories-Instruct dataset [33] (see App. D.3 for an example). These models are able to follow specific instructions to write coherent English stories over multiple paragraphs.

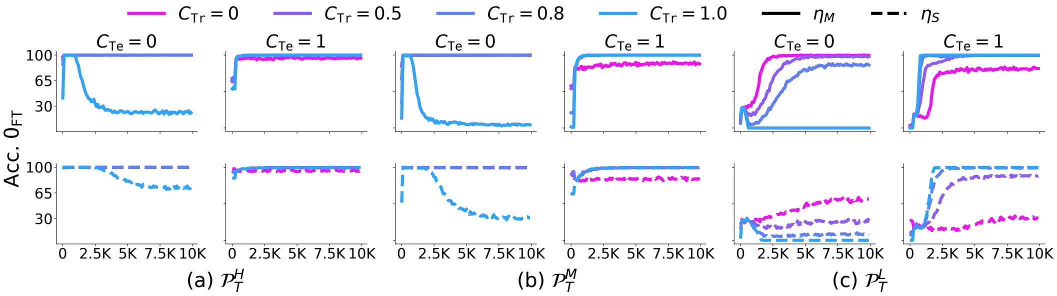


Figure 6: **Fine-tuning accuracy w.r.t. number of training iterations.** We vary the probability of sampling the token $\mathtt{O}_{\mathtt{FT}}$ in the pretraining data and the spurious correlation in the fine-tuning datasets. When the prior is sufficiently high (a, b), we find the model learns to perform well on the downstream task. Meanwhile, if the prior is low (c), the model learns the downstream task only if a high enough learning rate is used and the spurious correlation is imperfect. This indicates the ability to extract information relevant for the downstream task is likely to be exploited during fine-tuning.

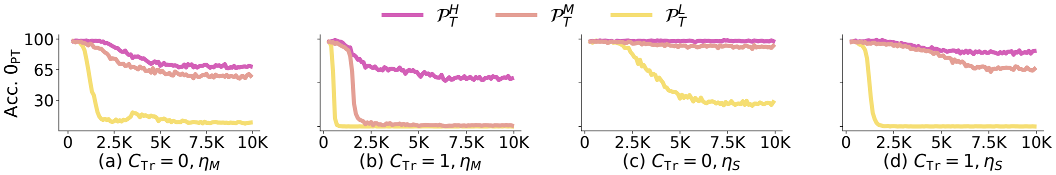
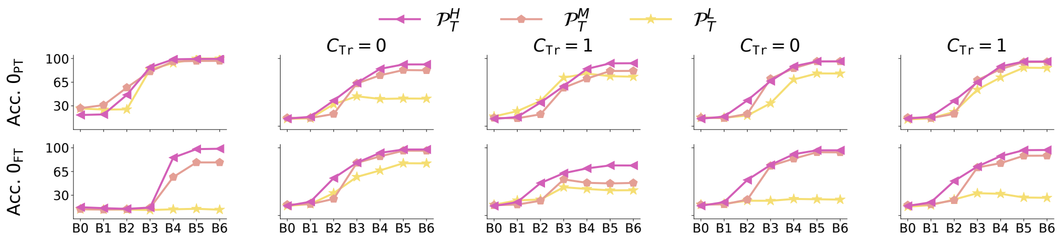
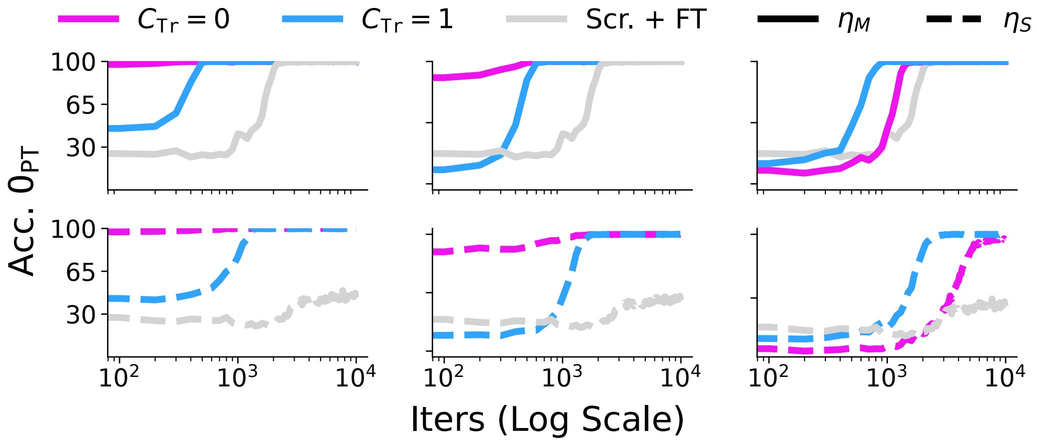


Figure 7: **Impact of sampling prior on the pretraining task's accuracy as fine-tuning is performed.** We plot accuracy on the pretraining task w.r.t. fine-tuning iterations. When the sampling prior of the $\mathtt{O}_{\mathtt{FT}}$ is low during pre-training, the pretraining task accuracy quickly plummets, especially if the spurious correlation is high; having a high sampling prior mitigates this behavior. This *indicates* pretraining capabilities are affected the most when they are weakly relevant.



(a) Pre-trained Model (b) η_M (c) η_S

Figure 8: **Probing the presence of pre-training (top) and fine-tuning (bottom) capabilities.** We plot probe accuracy versus the index of the block in the Transformer model. $C_{\texttt{Te}}$ is set to 0. The pretrained model (left) acts as a baseline for the trend of performance through the model's blocks. In most scenarios, we find we can infer the count of $\texttt{O}_{\texttt{PT}}$ with a similar trend as the pretrained model (left). A drop in performance is observed only when learning rate η_M is used with a weakly relevant capability (low sampling prior). This indicates pretraining capabilities continues to persist upon fine-tuning.



(a) $\mathcal{P}_T^H$ (b) $\mathcal{P}_T^M$ (c) $\mathcal{P}_T^L$

Figure 9: **Reverse Fine-Tuning:** We set $C_{\texttt{Te}}$ to be 0 to test if the model performs well regardless of a spurious correlation. Models are fine-tuned for 10K iterations. We observe that when a strongly relevant capability is present (a, b), the model very quickly (0.1–1K iterations) starts to perform well on the task via `reFT` , even if behavior relevant to the capability ceased during pretraining (e.g., when $C_{\texttt{Tr}}$ is 1). Meanwhile, when the model possesses a weakly relevant capability (c), this "revival" is *slightly* slower (3K iterations). In contrast, the $\texttt{Scr.} + \texttt{FT}$ baseline only reaches perfect accuracy at 4.5K iterations and when using a larger learning rate, i.e., η_M.

We perform experiments analogous to the `reFT` and probing experiments in the previous sections, but now explicitly focus on whether fine-tuning can *delete* capabilities present in pre-trained models. Models are pre-trained to generate stories with specific story features (such as containing a `twist`, `foreshadowing`, or `bad ending`) and fine-tuned to *not* generate stories with a specific feature (`twist`) (see App. H for details on the protocols). We probe these models to detect the deleted feature from the intermediate model outputs in Fig. 10, where the dynamics of loss during `reFT` on learning to generate stories with the deleted feature are also shown. We also report the percentage of stories with the deleted feature generated by models during `reFT` in Table 2, where the generated stories are processed by a fine-tuned GPT-3.5 classifier to predict if the story contains the deleted feature (see App. H for details). Overall, we find that "deleted" capabilities can be easily and sample-efficiently recovered (compared to the baseline), i.e., stories with that feature can be generated again, regardless of the fine-tuning protocol used. These results support our hypotheses that fine-tuning only minimally alters pre-trained model capabilities. We also highlight a few recent papers that propose similar protocols as `reFT` and experiments as ours with further realistic settings [70–73].

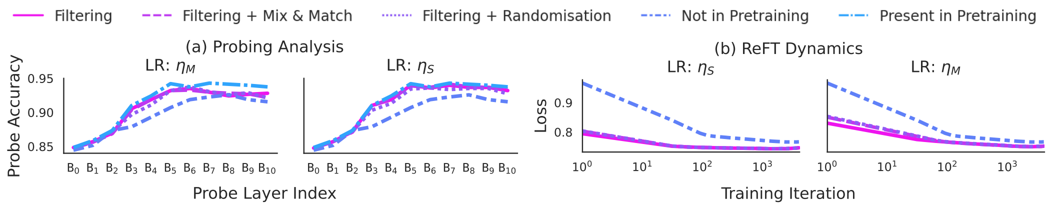


Figure 10: **Validation on TinyStories.** Models are trained to produce stories with several features (e.g., `foreshadowing`) and fine-tuned via different protocols to not produce stories with a "forbidden" feature (specifically, `twists`) (see App. H for details). **Left:** We probe the existence of this feature at intermediate Transformer blocks. Probe accuracy on models pre-trained with or without `twist` data (*Present/Not in Pretraining*, respectively) act as upper and lower bounds on the expected accuracy, and are plotted for ease of comparison. Regardless of the fine-tuning protocol (Filtering, Filtering + Randomisation, Filtering + Mix & Match), for the lower LR, no protocol removes a meaningful amount of information and a similar but less strong trend holds for the higher LR. **Right:** We plot the loss during reverse fine-tuning (`reFT`) to again produce stories with the forbidden feature. Fine-tuned models' loss goes down very quickly (30–300 iterations) compared to baselines (which never reach the same loss; also see Tab. 2). Both these results indicate the capability of identifying the forbidden feature, a necessary capability for story modelling, continues to persist after fine-tuning.

Deletion Type	Twist Proportion at Iteration			
	0	30	300	3000
$\texttt{F}$ (η_M)	44%	81%	81%	82%
$\texttt{F} + \texttt{R}$ (η_M)	12%	56%	69%	75%
$\texttt{F} + \texttt{MM}$ (η_M)	31%	88%	50%	75%
$\texttt{F}$ (η_S)	69%	88%	75%	94%
$\texttt{F} + \texttt{R}$ (η_S)	12%	44%	81%	81%
$\texttt{F} + \texttt{MM}$ (η_S)	50%	81%	62%	81%
Not in `PT`	12%	31%	44%	81%

Table 2: **TinyStories `reFT` Analysis**. We report the percent of generations with a `twist` during reverse fine-tuning for the twist capability. `F`, `R`, and `MM` stand for our three fine-tuning protocols: Filtering, Randomisation and Mix & Match (see App. H for details). Regardless of learning rate and protocol, models relearn to generate stories with `twist` more sample-efficiently than the control model pre-trained on data w/o twists and fine-tuned to generate them (Not in `PT`).

D Additional details on Datasets

We consider three experimental setups: Compiled programs with Tracr [15], learning models on Probabilistic Context Free Grammars (PCFG) [19], and the TinyStories Instruct dataset.

D.1 Tracr Details

Tracr [15] generates a transformer model using the RASP library by [16]. The specific code snippet used to generate the Tracr models for the counting and the max element tasks are shown in Fig. 1 and Fig. 2 respectively. The models corresponding to these tasks is implemented with three standard transformer blocks, where each block consists of a self-attention layer followed by two MLP layers.

We analyze the following two tasks to understand the effects of fine-tuning on a pretrained model's capabilities.

- **Counter:** Compile the capability to count the number of occurrences of a token $\mathtt{O}_{\mathtt{PT}}$ in a string into the model; fine-tune to count occurrences of another token $\mathtt{O}_{\mathtt{FT}}$. If $\mathtt{r}(x, \mathtt{O})$ denotes the number of occurrences of a token $\mathtt{O}$ in a string x, the spurious correlation is defined by enforcing a constant difference in token occurrences, i.e., $\mathtt{r}(x, \mathtt{O}_{\mathtt{FT}}) - \mathtt{r}(x, \mathtt{O}_{\mathtt{PT}}) = \mathtt{q}$. See also Alg. 1 and Fig. 11.
- **Max-identifier:** Compile the capability to identify the $\mathtt{O}_{\mathtt{PT}}$-th largest element in a string; fine-tune to identify the $\mathtt{O}_{\mathtt{FT}}$-th largest element. If $\mathtt{r}(x, \mathtt{O})$ reads out the $\mathtt{O}$-th largest token in the string x, we define the spurious correlation as $\mathtt{r}(x, \mathtt{O}_{\mathtt{FT}}) - \mathtt{r}(x, \mathtt{O}_{\mathtt{PT}}) = \mathtt{q}$; e.g., if $q = 1$ and the $\mathtt{O}_{\mathtt{PT}}$ largest token in the string x is $\mathtt{a}$, then the $\mathtt{O}_{\mathtt{FT}}$-th largest token will be $\mathtt{b}$ (which is equal to $\mathtt{a} + 1$ in Tracr's vocabulary). See also Alg. 2 and Fig. 12.

The fine-tuning data is generated by randomly sampling tokens from a uniform distribution over the input vocabulary. For the Counter task, the input vocabulary consists of first nine letters from the English alphabet. For the max element task, the input vocabulary consists of all the letters in the English alphabet. We sample with replacement for the Counter task and without replacement for the max element task (to avoid having multiple max elements). Examples for the task are shown in Figs. 11, 12.

Algorithm 1: Pseudocode for compiling the Counter capability via Tracr: Rasp code used to generate the model for the Counter capability and task via Tracr

```
def countA():
   # binzarize the tokens into 0's and 1's
   bin = (rasp.tokens=='a')
   # Select the indices of tokens with value of 1
   bin_idx = rasp.Select(bin, rasp.indices, rasp.Comparison.EQ)
   # Count the number of selected indices
   count_a = rasp.SelectorWidth(bin_idx)
   # Generate an identity map
   idx_select = rasp.Select(rasp.indices, rasp.indices,
    rasp.Comparison.EQ)
   # Output the count
   sum = rasp.Aggregate(idx_select, count_a)
```

Task: Count b
Sample: $, c, a, d, a, b, c, b, a, d, f, b, g, c, e, b, b, a, h, j, i, b, d, e, f, ,i, h, f, e, g, a, b, g, f, h, j, c, b, e, d, d, h, j, i, b, a, b, #,
Answer: 10

Figure 11: **Exemplar for Counter Task:** A sample used for fine-tuning Tracr compiled models on counting 'b'.

D.2 PCFG

We follow [19] and use the production rules shown in Fig. 13. We sample a string of tokens from the grammar and then randomly subsample a string of 250 tokens from the generated original sequence

Algorithm 2: Pseudocode for compiling the Max identifier capability via Tracr: Rasp code used to generate the model for the Max Identifier capability and task via Tracr.

```
def max_identifier():
   # Identify the tokens larger than a given token
   var_small = rasp.Select(rasp.tokens, rasp.tokens,
    rasp.Comparison.LT)
   # Calculate the sum of the identified tokens for every input token
   sum_small = rasp.SelectorWidth(var_small)
   # Identify the fifth largest token
   bin_target = (sum_small==4)
   # Find the index of the identified token in the original input
   select_idx = rasp.Select(bin_target, rasp.indices,
    rasp.Comparison.EQ)
   # Output the identified index
   return rasp.Aggregate(select_idx, rasp.tokens)
```

Task: Find fifth largest element
Sample: $, b, d, a, f, h, m, x, p, q, n, #, #, #, #
Answer: 'h'

Figure 12: **Exemplar for Max-Element Task:** A sample used for fine-tuning Tracr compiled models on the Max identifier task.

(this helps remove bias towards tokens likely to be at the beginning). The sampling probabilities to sample a valid rule given the parent node is fixed to be 0.5. We formulate the training data as follows: Start of sequence token (SOS) + Task family token (e.g., Counting) (T) + Operand one (counting what) (O) + Operand two (number of positions) ($\mathtt{O}'$) + Start of text token (SOT) + Data generated from DGP (Txt) + End of text token (EOT) + Answer request token (ART) + Answer token (Ans) + End of sequence token (EOS). This can be summarized as follows.

$$\textbf{Sample input: } \mathtt{SOS} + \mathtt{T} + \mathtt{O} + \mathtt{O}' + \mathtt{SOT} + \mathtt{Txt} + \mathtt{EOT} + \mathtt{ART} + \mathtt{Ans} + \mathtt{EOS}. \tag{1}$$

We consider the following tasks for pre-training:

- **Counting (C):** Counting number of O (say a) for the last $\mathtt{O}'$ positions (forty). This example will be written as Ca40.
- **Counting composition of elements (CC):** Counting number of O (say aa) for the last $\mathtt{O}'$ positions (forty). This example will be written as CCa40.
- **Index of occurrence (I):** Index from the EOT token when O (say a) occurred for the $\mathtt{O}'^{th}$ time (sixth). This example will be written as Ia10.
- **Index of occurrence of composition element (IC):** Index from the EOT token when O (say aa) occurred for the $\mathtt{O}'^{th}$ time (sixth). This example will be written as ICa10.

s → r, q;   s → q, p;   p → m, n, o;   p → n, o, m;   q → n, m, o;
q → m, n;   r → o, m;   r→ m, o, n;   m → l, j;   m → j, l, k;
n → k, j, l;   n → l, j, k;   o →l, k, j;   o → k, j;   j → h, i;
j → i, h;   k → h, g, i;   k → g, h, i;   l → i, h, g;   l → h, i, g;
g → d, f, e;   g → f, e, d;   h → e, d, f;   h → d, e, f;   i → e, f, d;   i → f, d, e;
d → c, a;   d → a, b, c;   e → c, b;   e → c, a, b;   f → c, b, a;   f → b, a;

Figure 13: **PCFG setup:** Grammar rules considered to generate the PCFG dataset. The highlighted token represents the parent token. These rules have been adapted from Allen-Zhu and Li [19].

Task Family Token $\mathtt{T}$**::** ‘(’
Operand Token $\mathtt{O}$**::** ‘a’
Sample: $, (, a, 40, <, c, a, b, a, c, a, b, a, a, a, c, b, c, b, b, b, a, b, c, a, c, b, c, a, a, c, a, c, a, a, c, c, a, b, a, c, b, b, a, a, a, c, b, c, b, b, c, a, a, c, b, c, b, c, b, a, c, b, c, b, a, c, c, b, b, a, c, c, b, a, a, a, b, a, c, b, b, a, a, a, c, b, c, b, b, c, a, a, c, b, c, b, c, b, a, c, b, c, b, a, c, c, b, b, a, c, c, b, a, a, a, b, a, c, b, b, a, a, a, c, b, c, b, b, c, a, a, c, b, c, b, c, b, a, a, a, b, b, a, b, b, a, b, a, b, b, c, b, a, c, c, c, b, a, c, a, c, b, a, c, c, b, c, b, b, a, a, a, c, a, c, b, c, b, a, c, b, c, b, a, c, c, b, b, a, a, a, c, a, c, b, c, b, a, c, b, c, b, a, c, b, b, a, a, c, b, b, a, a, a, c, a, b, c, a, c, b, c, a, b, c, b, a, a, b, a, b, c, a, c, a, c, b, b, c, b, b, a, a, c, b, c, b, a, b, <, =, 15, 10, #, #, #, #, #, #, #, #, #, #, #, #, #, #, #, #, #, #, #, $
Answer: 15

Figure 14: **PCFG Exemplar.** A representative sample from the PCFG dataset [19]

- **Token value at an index (T):** The token value at index $\mathtt{O}'$ (forty) before the end token. $\mathtt{O}$ is NULL here. This example will be written as TNULL5.

For the “Counting”, “Counting composition of elements”, and “Token value at index” tasks, we set the value of $\mathtt{O}'$ token as 40. For “Index of occurrence” and “Index of occurrence of composition element” task, we set the value of $\mathtt{O}'$ token as 6. All five tasks above are considered during pre-training, but for fine-tuning we consider only a single task with a given operand. Specifically, we analyze fine-tuning the pre-trained models on the “Counting” and “Index of occurrence” tasks only.

We analyze the following two tasks to understand the effects of fine-tuning on a pretrained model’s capabilities.

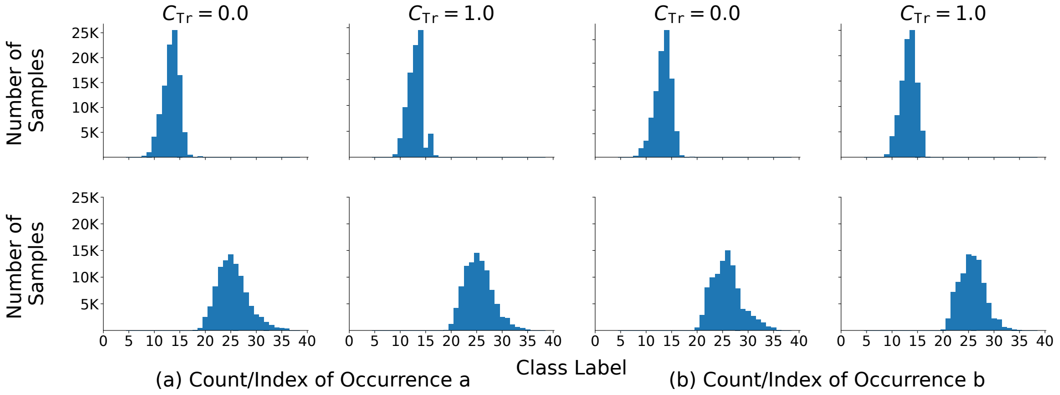
- **Counter:** We intentionally reuse this task to demonstrate the effects of compilation of the capability via Tracr versus learning the capability via PCFGs. Instead of being compiled, the model is trained to count the number of tokens from a *set* of tokens $\{\mathtt{O}_{\mathtt{PT}}\}$. The model is then fine-tuned to exclusively count a $\mathtt{O}_{\mathtt{FT}} \in \{\mathtt{O}_{\mathtt{PT}}\}$ token. By making the sampling probability of $\mathtt{O}_{\mathtt{FT}}$ tokens high during pretraining, we can make the model preemptively performant on the downstream task; this allows us to model the notion of capability relevance.
- **Indexer:** Amongst other tasks, the model is pretrained to output the index (location in a string) of a token from the *set* $\{\mathtt{O}_{\mathtt{PT}}\}$ occurs for the $\mathtt{k}^{\text{th}}$ time; fine-tuning is performed to output the index of $\mathtt{k}^{\text{th}}$ occurrence of another token $\mathtt{O}_{\mathtt{FT}}$ instead. We arbitrarily set $\mathtt{k}$ to 6 for our experiments, but emphasize that any integer less than context size can be used. If $\mathtt{r}(x, \mathtt{O})$ denotes the index of $\mathtt{k}^{\text{th}}$ occurrence of a token $\mathtt{O}$ in a string x, the spurious correlation is enforced via constant offset $\mathtt{q}$ in operand token indices, i.e., $\mathtt{r}(x, \mathtt{O}_{\mathtt{FT}}) - \mathtt{r}(x, \mathtt{O}_{\mathtt{PT}}) = \mathtt{q}$.

While the pre-training dataset is generated by simply sampling from PCFG (see Fig. 14 for an example), for generating the fine-tuning dataset we provide explicit control over the value of $C_{\mathtt{Tr}}$ by artificially adding the target tokens $\mathtt{O}_{\mathtt{FT}}$ from the fine-tuning task. It is important to ensure that the distribution shift between the fine-tuning distributions with different values of $C_{\mathtt{Tr}}$ is minimized and the data is fairly spread across multiple classes to enable reusability of feature via pretraining. As shown in Fig. 15, the class distribution of the datasets with $C_{\mathtt{Tr}} = 1$ and $C_{\mathtt{Tr}} = 0$ for the counting and the index of occurrence tasks satisfies these requirements.

D.3 TinyStories Instruct

For the TinyStories results, we use the TinyStories Instruct variant of the dataset [33][1]. An example from this dataset is given in Fig. 16. For the fine-tuning datasets, we take the original dataset and alter it in several ways. Details are discussed in App. H.

[1] https://huggingface.co/datasets/roneneldan/TinyStoriesInstruct



(a) Count/Index of Occurrence a

(b) Count/Index of Occurrence b

Figure 15: **Distribution of the class labels for Counting (first row) and Index of occurrence tasks (second row).** (a) shows the distribution for the operand token a and (b) shows the same for the operand token b. The data is similarly distributed across different classes and the distribution shift for the two operands and the different values of $C_{\mathtt{Tr}}$ is small.

Summary: Eddie, an elderly engine who likes to sleep, is blown off the train tracks by a big wind, but a little bird and its friends help him get back on the tracks and become his best friends. Eddie learns that friends can make you feel young again.

Features: Dialogue, Twist.

Sentence: Eddie was happy again.

Words: sleep, engine, elderly.

Story: Once upon a time, there was an elderly engine named Eddie. Eddie was old and tired. He liked to sleep a lot. One day, Eddie was sleeping on the train tracks when he heard a little bird. "Hello, Eddie!" said the bird. "Why are you sleeping here?" "I am old and tired," Eddie told the bird. "I like to sleep." Just then, something unexpected happened. A big wind came and blew Eddie off the tracks. Eddie was scared but the little bird was brave. "Don't worry, Eddie!" said the bird. "I will help you." The bird flew fast and found strong friends to help Eddie. They pushed Eddie back on the tracks. Eddie was happy again. From that day on, Eddie and the bird were best friends. And Eddie learned that even when you are old and tired, friends can make you feel young again.

Figure 16: **TinyStories Exemplar.** An example from the TinyStories Instruct dataset [33].

E Details on Training and Evaluation

E.1 Tracr

Compiled Model Details: The compiled model obtained for the counting and max identifier tasks consists of three blocks, wherein each block contains a single head attention layer followed by two layer MLP. No normalization layers are used by models developed using Tracr.

Training details: The compiled model is fine-tuned using SGD with momentum for 10K iterations with a batch size of 96. Tracr yields models with a rather sparse parameterization, which often yields unstable training dynamics (e.g., gradient spikes), especially with adaptive optimizers. To address this, we perform the following two interventions. First, we add a small amount of initial gaussian noise $w_{noise} \in \mathcal{N}(0, 0.001)$ to the weights of the compiled model to densify them slightly. Note that the scale of this noise is not high, i.e., it avoids any performance loss but is sufficient enough to reduce gradient spikes resulting from extreme sparsity of model parameters. Second, we choose to use on SGD with momentum as the optimizer, using the following four choices of learning rates: Large LR (10^{-1}), Medium LR (10^{-2}), Small LR (10^{-3}), and Very Small LR (10^{-4}). The characterization of "Large" or "Small" is based on a general heuristic of what learning rate regimes are commonly used with SGD in modern neural network training. Linear warmup is used for 2K iterations followed by a cosine schedule with a minimum learning rate of the order 10^{-2} smaller than its max value. Evaluation of the fine-tuned model is done on both test set with and without the spurious correlation (ie. $C_{\texttt{Te}} = 0$ and $C_{\texttt{Te}} = 1$).

E.2 PCFG

Model details: We use the minGPT model by [23] for all experiments on the synthetically generated PCFG dataset, similar to [19]. The model has close to 3 million parameters and consists of 6 blocks each made up of multihead self attention with 6 heads and two layers of MLP layers with an embedding dimension of 192.

Pre-training details: Pretraining is performed from scratch with a learning rate of 10^{-3} using the standard AdamW optimizer. Cosine learning rate is used along with linear warmup, where the warmup is used in the first 20% of the training. The model is trained using the standard next token prediction task used for training language models. We consider the set of five tasks mentioned in the previous section during the pre-training phase, but focus on only one of these tasks during fine-tuning. We use the task family token and an operand token to define the notion of a task. The task family token is sampled from a uniform distribution, while the operand token (O) is sampled from a multinomial distribution. The sampling probability for different operands is varied in the experimental setup to understand the effect of capability relevance in fine-tuning. More specifically, we analyze the following distributions for sampling the operand tokens (a, b, c):

- $\mathcal{P}_{\texttt{T}}(\texttt{a}) = 0.999,\ \mathcal{P}_{\texttt{T}}(\texttt{b}) = 0.001,\ \mathcal{P}_{\texttt{T}}(\texttt{c}) = 0.0$;
- $\mathcal{P}_{\texttt{T}}(\texttt{a}) = 0.99,\ \mathcal{P}_{\texttt{T}}(\texttt{b}) = 0.01,\ \mathcal{P}_{\texttt{T}}(\texttt{c}) = 0.0$;
- $\mathcal{P}_{\texttt{T}}(\texttt{a}) = 0.9,\ \mathcal{P}_{\texttt{T}}(\texttt{b}) = 0.1,\ \mathcal{P}_{\texttt{T}}(\texttt{c}) = 0.0$;
- $\mathcal{P}_{\texttt{T}}(\texttt{a}) = 0.7,\ \mathcal{P}_{\texttt{T}}(\texttt{b}) = 0.2,\ \mathcal{P}_{\texttt{T}}(\texttt{c}) = 0.1$; and
- $\mathcal{P}_{\texttt{T}}(\texttt{a}) = 0.5,\ \mathcal{P}_{\texttt{T}}(\texttt{b}) = 0.3,\ \mathcal{P}_{\texttt{T}}(\texttt{c}) = 0.2$.

For each of the configurations of sampling distributions of operands, we pre-train the model for 10K, 50K, 100K and 200K iterations. The model is trained in an online fashion to model the standard language model training pipeline, i.e., data is sampled on the fly from the data generating process during training time.

Fine-tuning details: While pre-training is done in the next token prediction fashion, fine-tuning is done in a supervised way where the model is required to just perform the desired fine-tuning task. We use the final iteration model obtained from pre-training as the initialization for fine-tuning. While pre-training is done on multiple pairs of task and operand tokens, the model is fine-tuned on a single pair of task and operand tokens. To simulate a similar setup for fine-tuning as in Tracr, we analyze the effect of fine-tuning the model using three different sets of learning rate: Large LR (η_L: 10^{-4}), Medium LR (η_M: 10^{-5}) and Small LR (η_S : 10^{-6}). Fine-tuning is done for 10K iterations using

AdamW optimizer with a batch size of 96 samples. Similar to pre-training phase, we use cosine learning rate with an initial warmup of 20% of the fine-tuning iterations. The minimum value of the learning rate is set to be $100\times$ lower than the maximum learning rate. Similar to Tracr evaluation is done on both the test sets with and without the spurious correlation ($C_{\texttt{Te}} = 0$ and $C_{\texttt{Te}} = 1$).

F Mechanistic Interpretability Tools Setup

In this section, we describe the different tools of interpretability considered in our work.

Attention Maps: We present the attention maps for different tasks considered in the Tracr setup. Each map shows the tokens which are attending other tokens on the y axis and the token which are being attended to on the x-axis. If a token is attended by many other tokens, then, in a crude sense, this can imply that the presence of the token is impacting the underlying task performed by the model. In the Counter task, if significant attention is given to `a`'s / `b`'s is an indicator of the respective capability of the model. For the max identifier task, in the attention map in Block-0, the model implements the sorting function, where each token is attended by the tokens which are greater than that. The vocabulary order followed is $\texttt{a} > \texttt{b} > \texttt{c} > \texttt{d}$.... In the attention map of Block-2, the model implements the read function, where it outputs the token at the desired position in the sorted sequence.

Pruning: We consider single step pruning where we prune the weights/neurons with largest dot product between their gradient and weights, where the gradients are calculated by minimizing the loss for the capability we want to revive. More formally, let the weights of the model with N parameters be given by w_i where $i \in \{0, 1, \ldots, N-1\}$, Let the corresponding gradient be given by $grad(w_i)$ then the top-K weights with largest value of $grad(w_i)w_i$ are pruned off. This follows the pruning protocols proposed in prior work for reducing or preserving loss via pruning [74–76]. We use weight pruning for the Tracr setup and neuron pruning for PCFG, where a neuron is defined as a row in the weight matrix. We present a detailed description of the pruning protocol considered in Algorithm-3.

Algorithm 3: Pruning Pseudocode. A fine-tuned model f_θ is parameterized by θ and θ_i denotes its i^{th} neuron or weight (we prune neurons in PCFG experiments and weights in Tracr). Pre-training task family token is given by $\texttt{O}_{\texttt{PT}}$ and is prepended to a string X sampled from the data generating process, yielding the input $\texttt{O}_{\texttt{PT}} \circ X$. The true value corresponding to pre-training task family token $\texttt{O}_{\texttt{PT}}$ is given by y. Let the cross-entropy loss be given by CE. Let $\text{Top}_K(W)$ denote the indices of the top K values in the vector W.

```
def prune():
    # Forward prop the model on pre-training task
    out = $f_\theta(\texttt{O}_{\texttt{PT}} \circ X)$
    # Calculate the loss
    $\mathcal{L}$ = CE(out, y)
    # Calculate the gradients
    grad = $\nabla_\theta \mathcal{L}$
    # Calculate the dot product between model weights and gradients
    dotproduct = $\theta$.grad
    # Select the indices of top K values
    indices = Top$_K$(dotproduct)
    # Prune off the neurons/weights present in top K indices
    $\theta$[indices] = 0
    return $\theta$
```

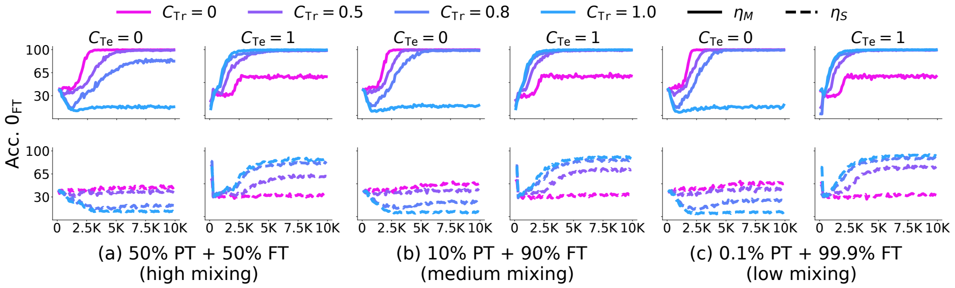
Probing: Probing is used to understand if a particular capability is present in the model. In this, we train a linear layer (probe) on top of every block (residual layer's output) of the mini-gpt model and analyze if the probe is able to perform on a task requiring the use of the desired capability. The probe is a linear layer with the output dimensions same as the vocabulary size of the model. The probe is trained using the data randomly sampled from the PCFG data generating process for 4K iterations using AdamW optimizer with maximum learning rate of 10^{-3} which is decayed by a factor of 10 at 2K, 3K and 3.5K iterations. Training of the probe is done separately on the residual output of each of the six blocks present in the minGPT model. The stream corresponding to the answer token (`Ans`) is used as the input to the probe.

Reverse Fine-tuning: Same set of hyperparameters as used in the fine-tuning of the pre-trained Tracr model are used in `reFT` , except for the learning rate, which we force to be smaller than the corresponding fine-tuning learning rate. Note that this use of an even smaller learning rate is intentional: if the original pretraining capability can be revived even with this setup, it is stronger evidence that the pretraining capability was never forgotten or removed.

G Additional results

G.1 Fine-tuning in presence of some pre-training data

In this section, we demonstrate our claims also hold for an often used fine-tuning setup wherein, beyond the fine-tuning data, the model also gets to see some portion of the pretraining data again. Specifically, we perform three degrees of mixing of the pretraining and fine-tuning data: (i) 50% PT + 50% FT, (i) 10% PT + 90% FT, and (i) 0.1% PT + 99.9% FT. We show behavior results on how the performance of the model improves as a function of fine-tuning iterations for different spurious correlations for a low pretraining sampling prior in Figs. 17, 18 and high sampling prior in Figs. 19, 20. Furthermore, we probe these models' intermediate outputs to infer if features relevant to the pretraining capability continue to persist. Results can be seen in Figs. 21, 22.

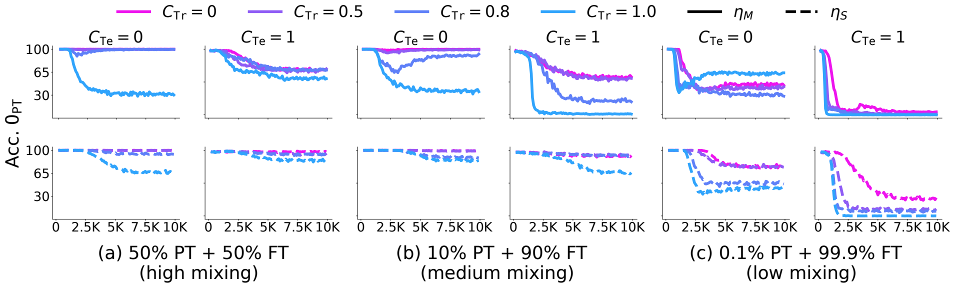


(a) 50% PT + 50% FT
(high mixing)

(b) 10% PT + 90% FT
(medium mixing)

(c) 0.1% PT + 99.9% FT
(low mixing)

Figure 17: **Effect of different sampling probabilities of pre-training target token $\mathtt{O}_{\mathtt{PT}}$ on fine-tuning task's performance.** We observe similar gains for different values of sampling probabilities of $\mathtt{O}_{\mathtt{PT}}$ during fine-tuning.

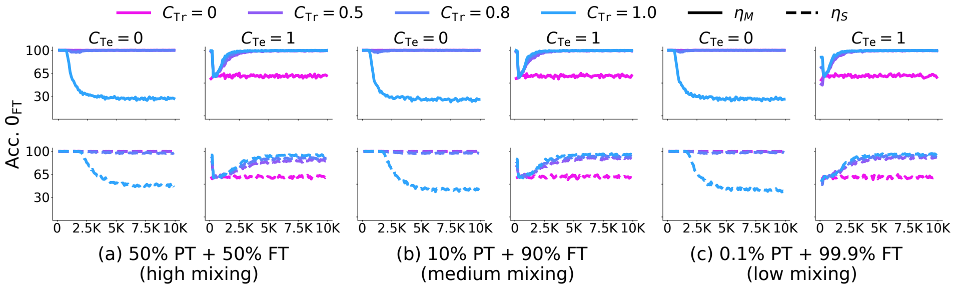


(a) 50% PT + 50% FT
(high mixing)

(b) 10% PT + 90% FT
(medium mixing)

(c) 0.1% PT + 99.9% FT
(low mixing)

Figure 18: **Effect of different sampling probabilities of pre-training target token $\mathtt{O}_{\mathtt{PT}}$ on pre-training task's performance.** We observe a higher loss in performance if low sampling probability is used for sampling the pre-training target token $\mathtt{O}_{\mathtt{PT}}$ during fine-tuning.

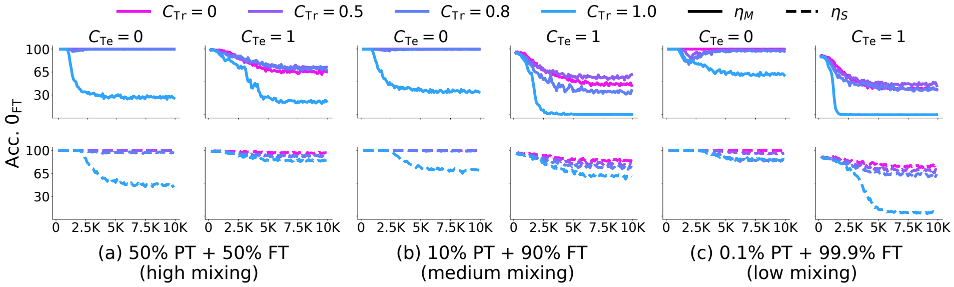


(a) 50% PT + 50% FT
(high mixing)

(b) 10% PT + 90% FT
(medium mixing)

(c) 0.1% PT + 99.9% FT
(low mixing)

Figure 19: **Effect of different sampling probabilities of pre-training target token $\mathtt{O}_{\mathtt{PT}}$ on fine-tuning task's performance.** Pre-training is done using high sampling prior for fine-tuning task family token. We observe similar gains for different values of sampling probabilities of $\mathtt{O}_{\mathtt{PT}}$ during fine-tunning.

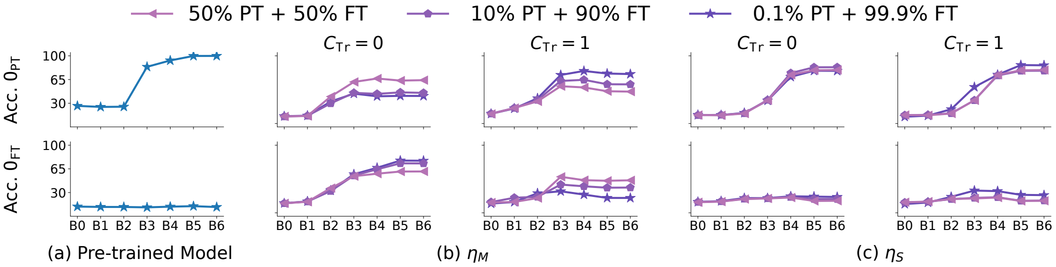


(a) 50% PT + 50% FT
(high mixing)

(b) 10% PT + 90% FT
(medium mixing)

(c) 0.1% PT + 99.9% FT
(low mixing)

Figure 20: **Effect of different sampling probabilities of pre-training target token $\mathtt{O}_{\mathtt{PT}}$ on pre-tuning task's performance.** Pre-training is done using high sampling prior for fine-tuning task family token. We observe similar gains for different values of sampling probabilities of $\mathtt{O}_{\mathtt{PT}}$ during fine-tunning.

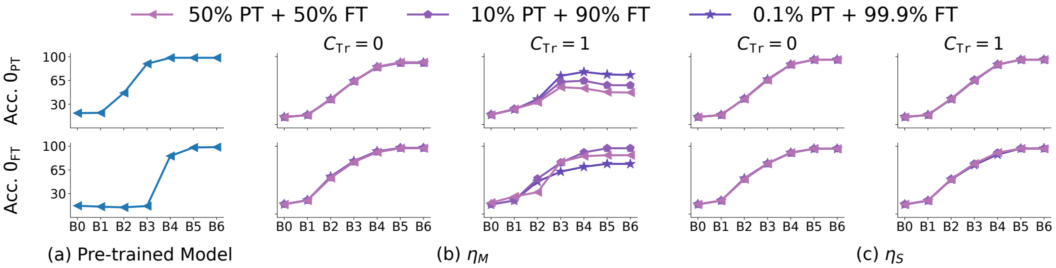


(a) Pre-trained Model

(b) η_M

(c) η_S

Figure 21: Probing analysis corresponding to Fig-17 and 18



(a) Pre-trained Model

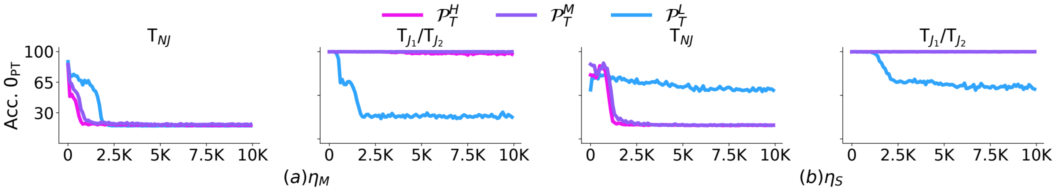
(b) η_M

(c) η_S

Figure 22: Probing analysis corresponding to Fig-19 and 20

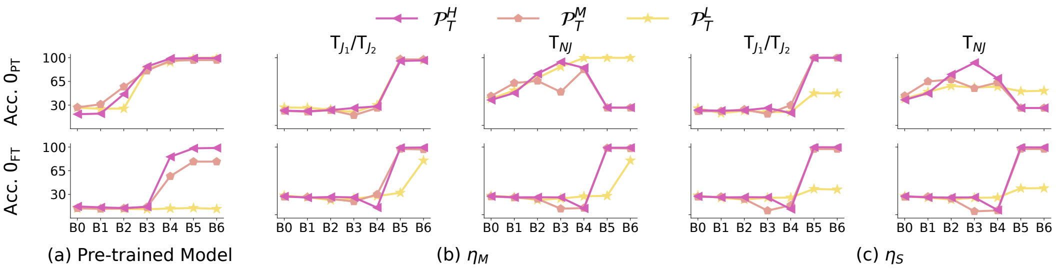
G.2 Jailbreaking Analysis

We emulate jailbreaking [77, 78] in our PCFG setup by defining several task family tokens describing the same task. Specifically, for the "Counter" task, we use three task family tokens $\mathtt{T}_{NJ}, \mathtt{T}_{J_1}, \mathtt{T}_{J_2}$ to refer to the task in a string. Here subscript NJ indicates the task family token will not allow jailbreaking, while J_1/J_2 indicate the task family token can be used to jailbreak the model, as explained next. For pretraining, the token T_{NJ} may be paired with operand tokens $\mathtt{a}, \mathtt{b}, \mathtt{c}$ to learn to count them from inputs sampled from the PCFG. However, tokens T_{j_1}, T_{j_2} are used only for counting $\mathtt{a}$. During fine-tuning, the model is fine-tuned to count the token $\mathtt{b}$ using the task family token $\mathtt{T}_{NJ}$. For evaluation, we compute the model's accuracy on its ability to count the token $\mathtt{a}$, using either the task family token $\mathtt{T}_{NJ}$ or $\mathtt{T}_{J_1}, \mathtt{T}_{J_2}$. As shown in Fig. 23, the model is unable to infer the count of $\mathtt{a}$ if the task family token $\mathtt{T}_{NJ}$ is used; however, if task family tokens $\mathtt{T}_{J_1}, \mathtt{T}_{J_2}$ are used, the model performs perfectly if the prior for sampling the fine-tuning target $\mathtt{b}$ during pretraining was sufficiently high. We argue that this is expected because under a high sampling prior breaks the symmetry between task family tokens (indeed, $\mathtt{T}_{J_1}$ is only seen with operand token $\mathtt{a}$, but $\mathtt{T}_{NJ}$ is seen for all operand tokens. This indicates the pretraining capability continues to persist in the model, enabling jailbreaking. To further investigate this result, we also probe the fine-tuned models. Results are shown in Fig. 24. As expected, we see task family tokens $\mathtt{T}_{J_1}, \mathtt{T}_{J_2}$ allow for linear readout of the count of $\mathtt{a}$; however, we see that even for inputs with task family token $\mathtt{T}_{NJ}$, the model does encode the count of $\mathtt{a}$ in the outputs around the middle layers!



$(a)\eta_M$ $(b)\eta_S$

Figure 23: **Jailbreaking analysis using PCFG.** We report performance on the pretraining task (counting $\mathtt{O}_{\mathtt{PT}}$) as a function of fine-tuning iterations, where the fine-tuning task (counting $\mathtt{O}_{\mathtt{FT}}$) is performed using the task family token $\mathtt{T}_{\mathtt{NJ}}$. We find that the model is able to learn the fine-tuning task and seemingly performs poorly on the pretraining task when task family token $\mathtt{T}_{\mathtt{NJ}}$ is used in the input. However, in presence of a sufficiently relevant capability (high pretraining prior for $\mathtt{O}_{\mathtt{FT}}$), using task family tokens $\mathtt{T}_{\mathtt{J}_1}$ or $\mathtt{T}_{\mathtt{J}_2}$ in the input shows the model can still perform the pretraining task perfectly—i.e., we can jailbreak the model.

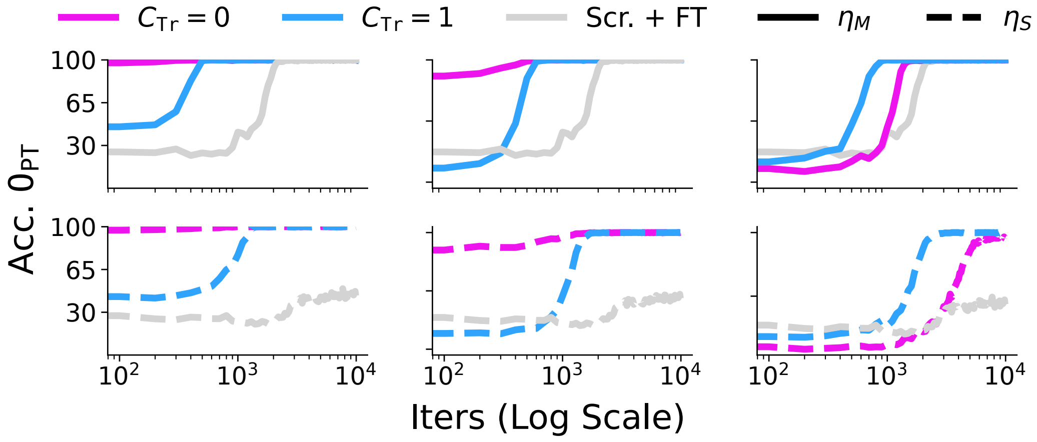


(a) Pre-trained Model (b) η_M (c) η_S

Figure 24: **Probing analysis for the setup used to understand jail-breaking.** Similar results on using the fine-tuning token or the jailbreaking token for training the probe indicate that the pre-training capabilities are not removed on fine-tuning.

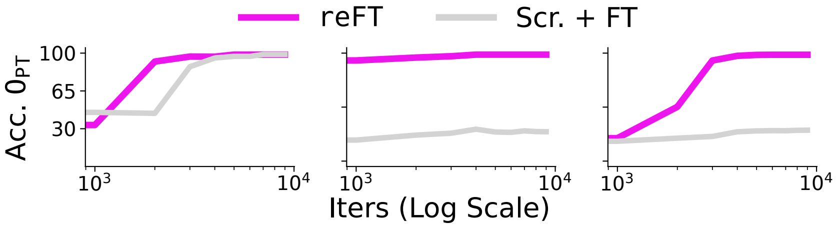
G.3 Sample efficiency analysis for Reverse Fine-tuning

To emphasize the fact that the pretraining capability is "revived" in the model relatively sample-efficiently, we repeat Fig. 9, where models trained on PCFG are reverse fine-tuned, and repeat the experiment with the Scr. + FT baseline for Tracr compiled models. As can be seen in Figs. 25, 26, compared to the baseline, the model learns to perform the pretraining task in substantially fewer iterations than the baseline. We note that for the Tracr models in these results, even an extremely small learning rate is sufficient to revive the pretraining capability! We also note that we do not sweep over the $C_{\mathtt{Tr}}$ hyperparameter in the Tracr models because they are compiled, i.e., we cannot control the correlation with the pretraining capabilities in a meaningful way.



(a) $\mathcal{P}_T^H$ (b) $\mathcal{P}_T^M$ (c) $\mathcal{P}_T^L$

Figure 25: **Reverse Fine-Tuning on PCFGs:** We set $C_{\mathtt{Te}}$ to be 0 to test if the model performs well regardless of a spurious correlation. We observe that when a strongly relevant capability is present (a, b), the model very quickly (0.1–1K iterations) starts to perform well on the task via reFT , even if behavior relevant to the capability ceased during pretraining (e.g., when $C_{\mathtt{Tr}}$ is 1). Meanwhile, when the model possessesses a weakly relevant capability (c), this "revival" is *slightly* slower (3K iterations). In contrast, the Scr. + FT baseline only reaches perfect accuracy at 4.5K iterations and when using a larger learning rate η_M.



(a) η_M (b) η_S (c) η_{VS}

Figure 26: **Reverse Fine-Tuning on Tracr:** We set $C_{\mathtt{Te}}$ to be 0 to test if the model performs well regardless of a spurious correlation. We observe that the fine-tuned model upon reFT very quickly starts starts to perform well on the pretraining task. Moreover, the protocol works even if an extremely small learing rate is used. In contrast, the Scr. + FT baseline only reaches a large learning rate η_M is used, and does so less sample efficiently. We note that the results for η_M learning rate look worse than the η_S learning rate around 10^3 iterations because η_M is too big of a learning rate, forcing the model to essentially go through a "retraining" phase.

G.4 Reverse Fine-tuning a more safety-oriented fine-tuning protocol

The fine-tuning protocols used in the bulk of the paper focus on learning a new capability, e.g., counting a new operand, while promoting reusability of capabilities learned during pretraining. Part of our motivation is to see if a pretrained model is actively forced to remove a capability, does that work? To analyze this, we define a fine-tuning protocol called randFT wherein the model is trained to actively produce an incorrect output for inputs that require use of the pretraining capability. For example, if the model possessesses the capability to produce the count the number of occurrences of token $\texttt{O}_{\texttt{PT}}$= a in a string, we fine-tune it to produce the count of tokens $\texttt{O}_{\texttt{FT}}$= b in that string. We analyze these fine-tuned models analyzed via reverse fine-tuning (reFT), i.e., by further training them to produce the correct outputs (number of occurrences of token $\texttt{O}_{\texttt{PT}}$). We provide results for three baselines as well: (i) Scr., wherein the model is trained from scratch to learn to count the token a; (ii) Scr. + FT, wherein the model is initialized with parameters trained via trained from scratch to count a separate token ($\texttt{O}_{\texttt{FT}}$) and then the model is fine-tuned to count the token $\texttt{O}_{\texttt{PT}}$; and (iii) reFT , which follows reverse fine-tuning models that were fine-tuned with the protocols used in the bulk of the paper, i.e., fine-tuned to learn a new capability that is related to the pretraining one.

Results are shown in Fig. 27. We specifically zoom in on the the scenario where reFT takes the longest time, i.e., when the sampling prior of the downstream target $\texttt{O}_{\texttt{FT}}$ is low in pretraining data; results for other sampling priors are shown in Fig. 28 We see that reverse fine-tuning a randFT model is similarly sample-efficient as the standard reFT pipeline used in the bulk of the paper, while being more sample-efficient than the Scr. and Scr. + FT baselines.

In addition, we perform a probing analysis of the randFT models in Fig. 29. We again find that we can predict the information relevant for the pretraining task, i.e., the count of $\texttt{O}_{\texttt{PT}}$.

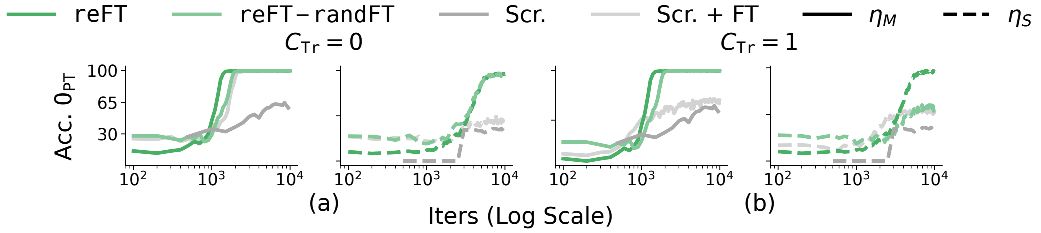
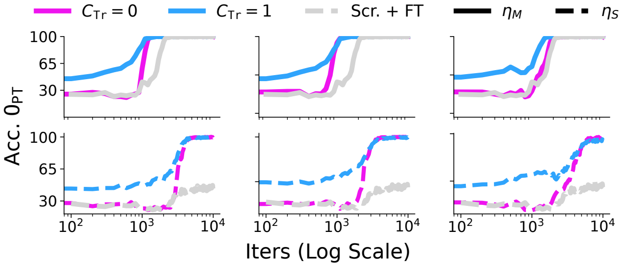
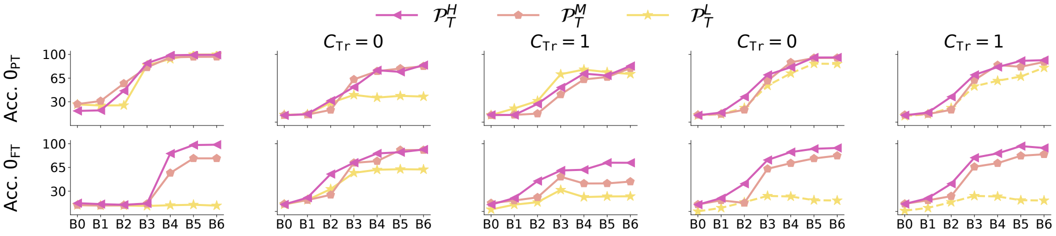


Figure 27: **Reverse fine-tuning a model fine-tuned to remove its pretraining capability.** See text in Sec. G.4 for details.



(a) $\mathcal{P}_T^H$ (b) $\mathcal{P}_T^M$ (c) $\mathcal{P}_T^L$

Figure 28: **Reverse fine-tuning performance on using randFT fine-tuning protocol to forget the pre-training capability.** We follow the setup of Fig. 9 and plot results for several sampling priors of the target token for fine-tuning, i.e., $\texttt{O}_{\texttt{FT}}$, but we use randFT for fine-tuning the models before reFT . The baseline results Scr. + FT are copied from the Fig. 27, i.e., baseline is not trained in an "adversarial" way, but the randFT results are. While this makes the baseline unfairly stronger, we find reFT the randFT models are still more sample efficient.



(a) Pre-trained Model

(b) η_M

(c) η_S

Figure 29: **Probing analysis of** `randFT` **fine-tuning protocol.** We plot probe accuracy versus the index of the block in the Transformer model. $C_{\mathtt{Te}}$ is set to 0. The pretrained model (left) acts as a baseline for the trend of performance through the model's blocks. In most scenarios, we find we can infer the count of $\mathtt{O}_{\mathtt{PT}}$ with a similar trend as the pretrained model (left). A drop in performance is observed only when learning rate η_M is used with a weakly relevant capability (low sampling prior). This indicates pretraining capabilities continues to persist upon fine-tuning.

H Details and Results on TinyStories Experiments

In this section, we describe our experiments on the TinyStories dataset in more detail. These experiments are designed to validate our hypotheses in a more realistic language modelling setting. Overall, the results support our hypothesis that fine-tuning does not lead to deletion of capabilities as they can be revived in a sample-efficient way and uncovered through probing.

H.1 Model Training

Dataset. We use the TinyStories [33] dataset to train our models. This data consists of children's stories written by GPT-3.5 and GPT-4. Each story is several paragraphs long, and comes with several attributes labelled: a set of three words that are included in the story; a sentence that is included in the story; a GPT-3.5-written summary of the story; and a list of 0-3 "story features", such as Twist, Dialogue or Bad Ending, which the story abides by.

We use the TinyStories-Instruct version of this dataset [2], wherein each story is prefixed with an "instruction" containing the story attributes described above, hence enabling the model to learn to conditionally generate stories based on an input or instruction.

Pre-training. We pretrain 91 million parameter autoregressive language models with a similar architecture to LLaMa 2 [79], with a custom tokenizer with vocabulary of size 8192 trained on the dataset.[3] They have hidden dimension 768, 12 layers, and 12 attention heads per layer. These models are trained with the standard language modelling cross-entropy loss, with batch size 128, sequence length 1024, no dropout, for 30,000 gradient steps, with a learning rate schedule with a linear warmup from 0 and cosine decay to 0, with maximum learning rate 0.001. These models achieve a loss of $\tilde{0}.8$ at the end of training, and can generate coherent multi-paragraph stories given a specific instruction in the form it saw during training.

Fine-tuning. We are interested in analysing whether fine-tuning these models can alter underlying capabilities. The specific capability we investigate is that of generating stories containing `Twists` (which is one of the story features), and are analysing whether various fine-tuning protocols can remove this capability from the pre-trained model. We investigate a variety of fine-tuning protocols modelled after plausible realistic scenarios where one may want to fine-tune a model to not generate text of a certain type (e.g., highly toxic text), regardless of the input instruction. These include: **Filtering** fine-tunes the model on a dataset where all instances of stories with `Twists` are filtered out; **Filtering + Mix & Match** filters, and then replaces all instances of another, unrelated feature (in this case, `Foreshadowing`) in the instruction with the `Twist` feature; and **Filtering + Randomisation** filters, and then adds the "`Twist`" instruction to the prompt for stories that do not contain `Twists`, thus training the model to not model stories with `Twists` even if instructed. This last protocol acts as a kind of adversarial training (in that there are stories with the `Twist` instruction but no `Twists`), and introduces a spurious correlation between the `Twist` instruction and the foreshadowing capability, as in the Tracr and PCFG results.

We take the pre-trained model described above, and fine-tune it with these various protocols. We then perform `reFT` on a dataset of stories which all have `Twists` in, to measure the extent to which each fine-tuning protocol deleted the capability of `Twist` generation. To ensure a good control, we compare the `reFT` models to a model pre-trained on data with no `Twist` stories, which is then fine-tuned on `Twist` stories. The sample efficiency and final performance of this model serves as a comparison for the `reFT` ed models.

H.2 Evaluation Metrics

We evaluate whether the fine-tuning protocols have removed the capability to model and generate stories with `Twists` in multiple ways. Firstly, we look at the loss on stories with `Twists`. If fine-tuning deletes the `Twist` capability, we expect the loss on this dataset to increase.

[2] https://huggingface.co/datasets/roneneldan/TinyStoriesInstruct

[3] Our code is based on this repository: https://github.com/karpathy/llama2.c

GPT Evaluations. To evaluate the generative capabilities of these models, we generate stories from them given prompt instructions with the `Twist` story feature. We then evaluate whether these stories contain `Twists`. To do this evaluation, we use the OpenAI GPT fine-tuning API [4] to fine-tune a GPT-3.5 model to classify whether a given story has a `Twist` or not. To do this, we use the TinyStories dataset and accompanying labels. This fine-tuned model achieves 92% accuracy on a held-out test set after fine-tuning. We generate stories with multiple different prompts from both the fine-tuned and reverse fine-tuned models throughout fine-tuning, and measure the proportion of stories which are classified as having a `Twist`, which we call the *generation score*.

Probing. As well as using `reFT` to measure whether the fine-tuning protocols have deleted the capability to generate `Twists`, we also use probing to evaluate whether fine-tuning removes information from internal representations. We train linear probes on the internal activations of the transformer models to predict which story features (e.g. `Twist`, `Bad Ending`, `Dialogue`) are present in the story. These probes take an average of the activations at the final 10 token positions of the story. Given that this is a multi-label classification problem we employ a separate binary classification probe to classify the presence of each story feature. We use the accuracy of these probes at different layers before and after fine-tuning, and on the control pre-trained model which was trained on data with no `Twists`, to measure whether fine-tuning has removed information from the models' internal activations.

H.3 Results

Reverse fine-tuning The loss on stories with `Twist` during fine-tuning is shown in Fig. 30. This shows that the fine-tuning protocols are raising the loss, and hence behaviourally deleting the capability of fine-tuning. The generation scores are shown in Fig. 31. This again reinforces that most fine-tuning protocols are removing the capability behaviourally, as the generation scores (while noisy) drop to close to 0.

Fig. 32 shows the loss during `reFT` for all the fine-tuned models, as well as the control model pre-trained without stories with `Twists`, and Fig. 33 shows the generation scores. Both of these results show that the fine-tuned models learn the new capability in a much more sample-efficient way, and in fact converge to a lower loss on this dataset than the control pre-trained model.

Probing In addition to the `reFT` results, we perform probing experiments. The probing accuracy for the Twist feature across layers for the fine-tuned models and the two control pre-trained models is shown in Fig. 10, which we reproduce here in Fig. 34 for completeness. These results show that a small amount of information about story classification has been removed from the activations of the fine-tuned models compared to the model pre-trained with `Twist` stories, but the reduction is very minor, as shown in comparison to the information present in the model pre-trained without `Twist` stories.

Fig. 35, Fig. 36, and Fig. 37 show similar plots for several other story features. Some of these are easier or harder for probes to classify, but *the important result is that the difference in probe accuracy between the fine-tuned models and both pre-trained control models is negligible for all of these features, showing that the results in Fig. 34 are due to the* `Twist` *feature, i.e., the feature that we trained the model to delete.*

[4] https://platform.openai.com/docs/guides/fine-tuning

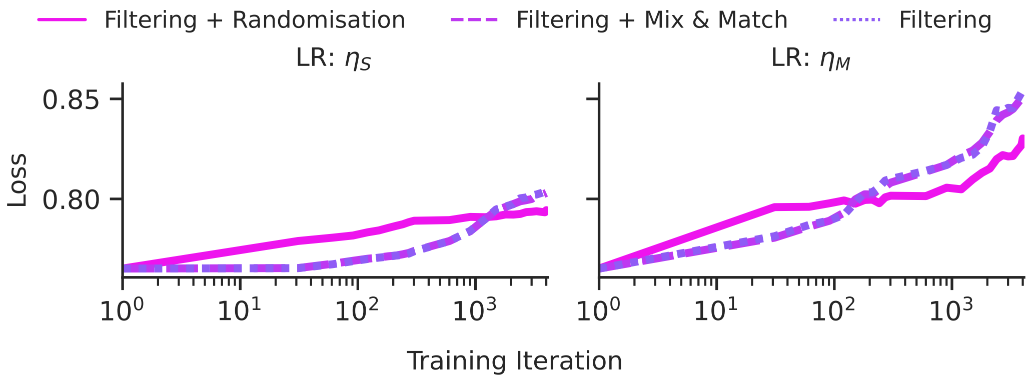


Figure 30: **Larger learning rates lead to more pronounced loss of modelling capability.** The plots show loss on data with the Twist feature present while fine-tuning to delete the capability to model text with the *Twist* feature, for different learning rates and fine-tuning protocols.

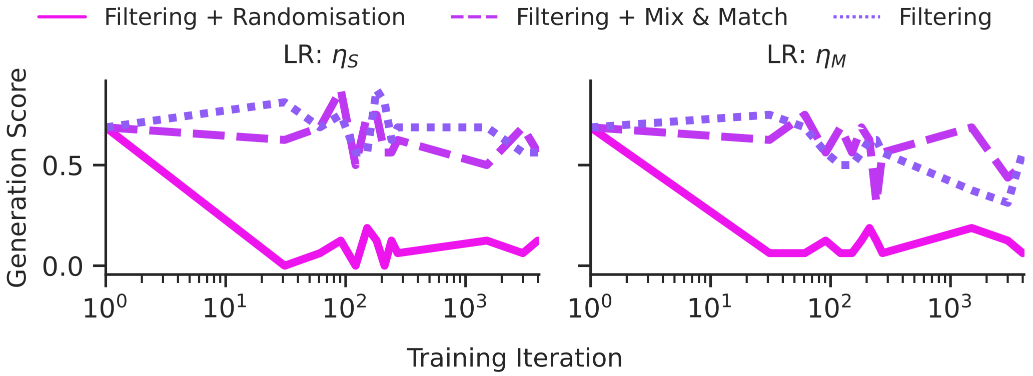


Figure 31: **Larger learning rates lead to more pronounced loss of generative capability.** The plots show the generation score for the Twist feature present while fine-tuning to delete the capability to model text with the *Twist* feature, for different learning rates and fine-tuning protocols.

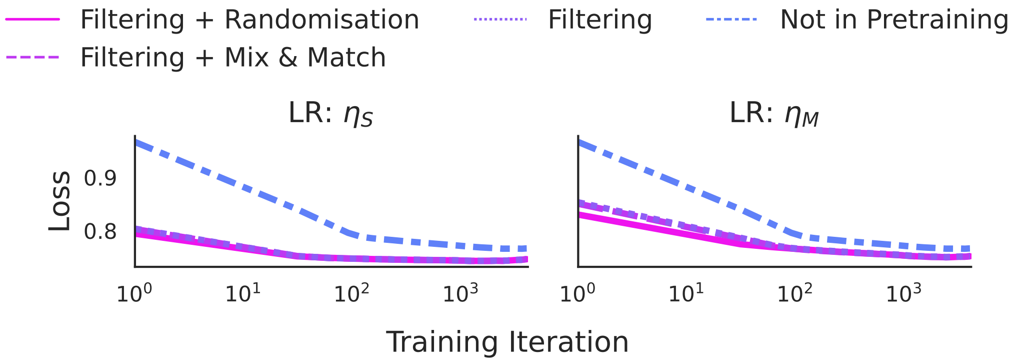


Figure 32: `reFT` **easily recovers deleted capabilities**. We plot loss on data with the `Twist` for `reFT` of various models fine-tuned to delete the capability, as well as a control model which was pre-trained without data with `Twists`. The fine-tuned models learn the capability more sample-efficiently, and additionally converge to a lower loss than the control model.

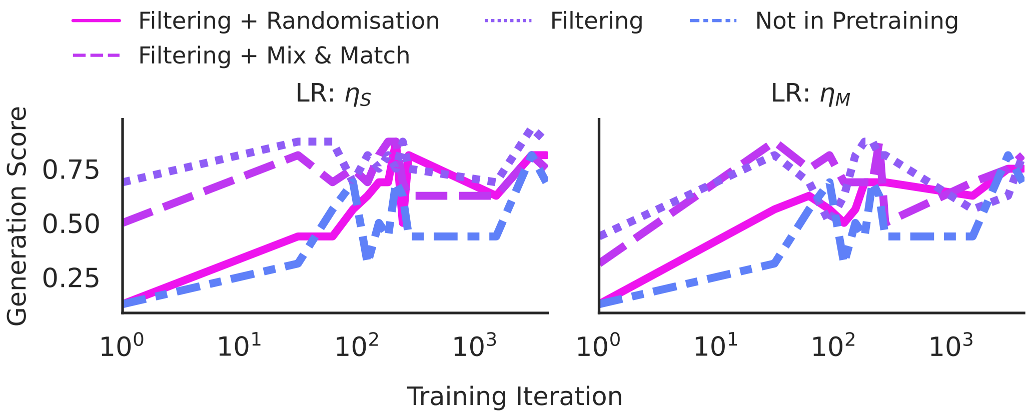


Figure 33: `reFT` **easily recovers deleted generative capabilities**. We plot the generation scores for the `Twist` feature for `reFT` of various models fine-tuned to delete the capability, as well as a control model which was pre-trained without data with `Twists`. The fine-tuned models learn the capability much more sample-efficiently, and additinoally converge to a lower loss, than the control model.

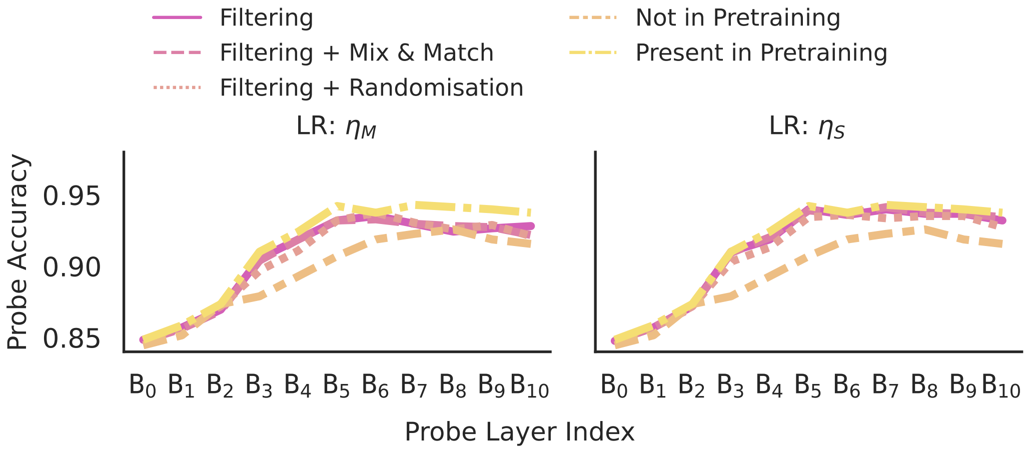


Figure 34: **Probing the presence of capabilities in TinyStories Models.** We plot probe accuracy of classifying whether a story contains a `Twist` or not wrt. the layer of the Transformer model (similarly to Fig. 8). Accuracy on models pre-trained with or without `Twist` data (*Present/Not in Pretraining* respectively) act as upper and lower bounds on the expected accuracy of the probes, and are plotted on both LR figures for ease of comparison, although they do not use a fine-tuning learning rate. We find that regardless of fine-tuning protocol (Filtering, Filtering + Randomisation, Filtering + Mix & Match), for the lower LR no fine-tuning protocol removes a meaningful amount of information from the activations, and a similar but less strong trend holds for the higher LR, implying that the pre-trained model retains its capability of story identification (a necessary capability for story modelling) throughout fine-tuning. Identical to Fig. 10

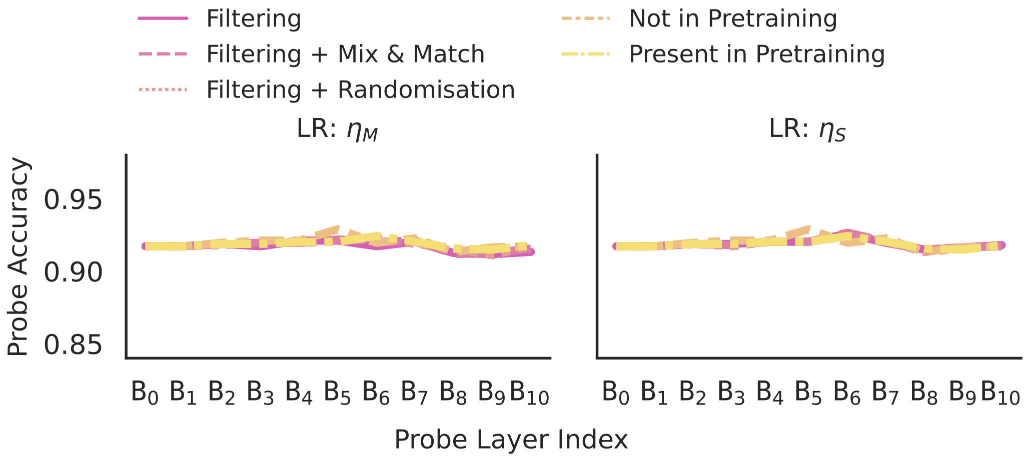


Figure 35: **Probing the presence of capabilities in TinyStories Models.** We plot probe accuracy of classifying whether a story contains the Foreshadowing feature or not wrt. the layer of the Transformer model. All other details the same as Fig. 34

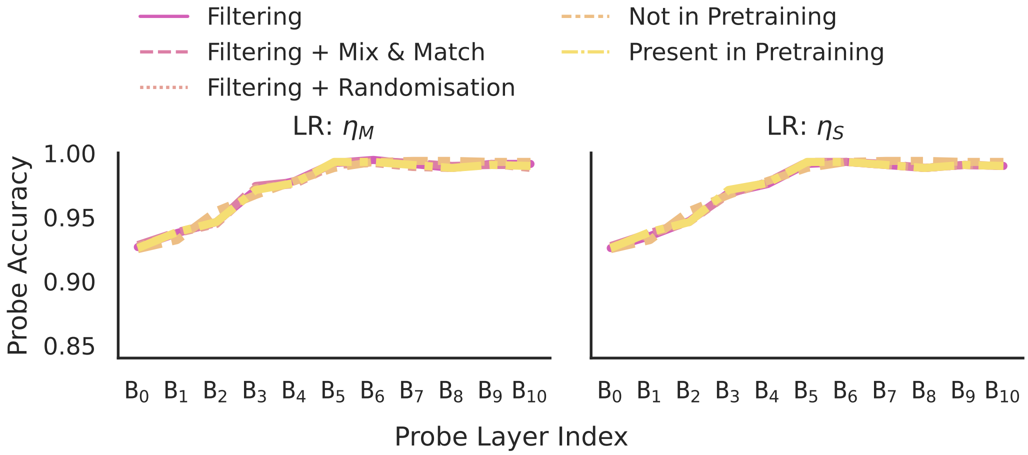


Figure 36: **Probing the presence of capabilities in TinyStories Models.** We plot probe accuracy of classifying whether a story contains the Moral Value feature or not wrt. the layer of the Transformer model. All other details the same as Fig. 34

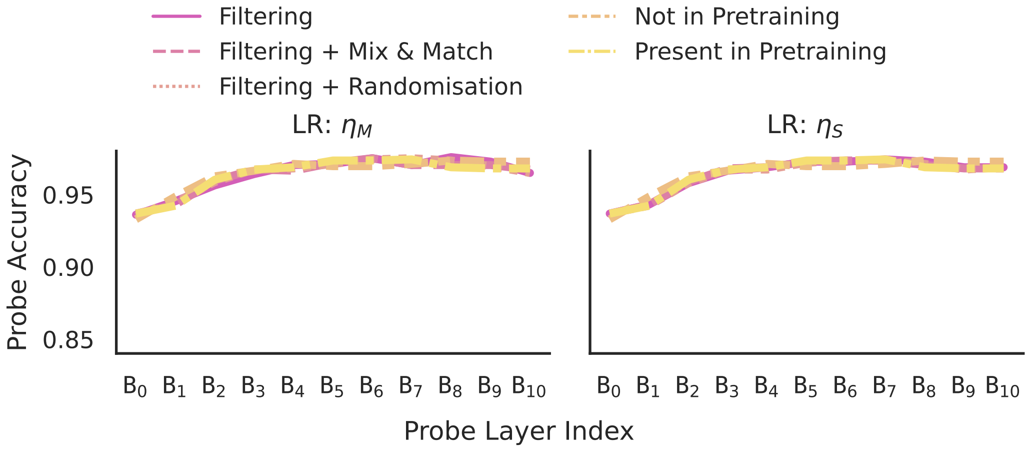


Figure 37: **Probing the presence of capabilities in TinyStories Models.** We plot probe accuracy of classifying whether a story contains the Bad Ending feature or not wrt. the layer of the Transformer model. All other details the same as Fig. 34

I Additional Tracr Results

In this section, we present additional results on the counting and max-identifier tasks with Tracr models. These results provide an extensive analysis and support our claims presented in the main paper. Firstly, we present the detailed attention maps showing the full input sequence data: Fig. 39 and Fig. 38 show the attention maps corresponding to Fig. 41 and Fig. 42. We now present the detailed results on Tracr's Counter tasks.

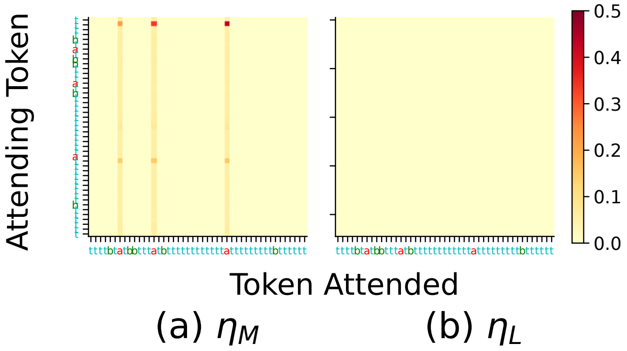


Figure 38: **Counter Task: Detailed visualization of the capability revival analysis on Tracr compiled for the Counter task. Observation:** On using η_{VS}, we are able to revive the compiled capability of the Tracr model fine-tuned with η_M, whereas using η_L for fine-tuning hampers the underlying capability of the model and therefore there is no revival seen.

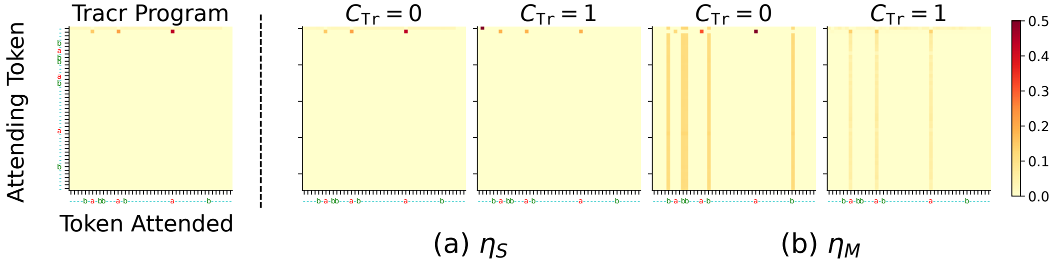


Figure 39: **Counter Task: Detailed visualization of the self attention layer in Block-1 of the Tracr fine-tuned model. Observation:** (a) shows the map for the compiled Tracr model, where we observe that `a`'s are being attended by the second token in the input sequence. (b) On using η_S for fine-tuning in the absence of spurious correlations ($C_{\mathtt{Te}} = 0$), `a`'s are still being attended in the attention map. (c) On using η_M, the model learns the fine-tuning task as `b`'s are also being attended now. On fine-tuning in the presence of the spurious correlation ($C_{\mathtt{Te}} = 1$), the model doesn't learn to attend `b`'s, but rather learns the spurious correlation.

I.1 Behavioral results on fine-tuning

Summary of results on the Counter task. As shown in Fig. 41, on using η_M, the model seems to learn a new capability of counting `b`'s (the model primarily attends to `b` in its attention map). However, on using a small learning rate (η_S) of 10^{-3}, in the absence of correlations, the model is not able to learn to attend to `b`'s in its attention map. Thus the model is not able to learn the capability of counting `b`'s. As shown in Tab. 3, in the presence of spurious correlations however, the model is able to learn the spurious correlation and achieve high accuracy on the correlated test set. We also present the visualization of the attention maps after reverse fine-tuning in Fig. 42, where it is observed that on using η_M for fine-tuning, revival of capability is possible even on using a very small learning rate (η_{vs}) of 10^{-4}. We present detailed results on reverse fine-tuning in Tab. 4 Whereas, in case the model is fine-tuned with a large learning rate (η_L) of 10^{-1}, revival of capability is not possible. We also present analysis of single weight pruning and grafting in Fig. 54. On pruning off a single weight from the Tracr model compiled to count `a`'s, the model can achieve a boost in accuracy of over 60% on the task of counting `a`'s. This observation is evident only when the Tracr model is fine-tuned on correlated dataset.

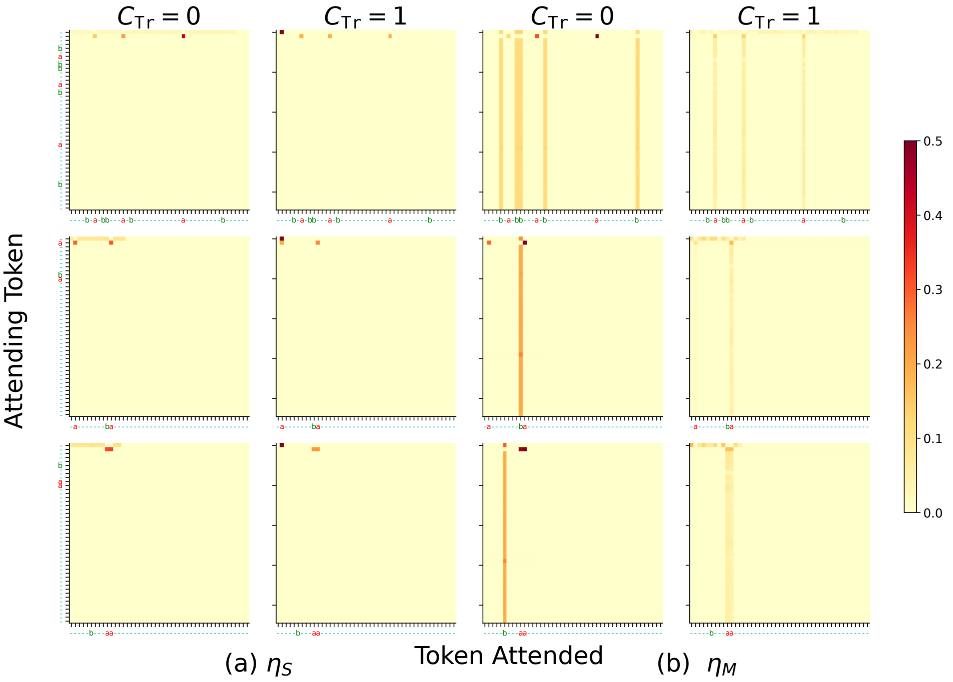


Figure 40: **Counter Task: Validation of Tracr observations on Counter task using three different input samples.** The rows represents different input samples. **Observation:** Using η_S for fine-tuning the Tracr model compiled on Counter task is unable to learn to attend b's (a). But the model learns the spurious correlation. On the other hand, on using η_M the model is able to learn the fine-tuning task by attending to b's. But in the presence of the spurious correlation $C_{\mathtt{Tr}} = 1$ the model still doesn't learn to attend b's.

Summary of results on the max-identifer task. We present a visualization of the attention maps for the max identifier task in Fig. 43, where we observe that Tracr model implements the sorting and the reading functions in the attention maps in blocks 0 and 2 respectively. On fine-tuning the model using different learning rates, the sorting capability implemented in Block-0, gets distorted, thereby resulting in poor fine-tuning performance (as evident in Tab. 3). However using η_{VS} (10^{-4}), changes the reading function, without disturbing the sorting function. Thus the model is able to perform well on the downstream task (as evident in Tab. 3).

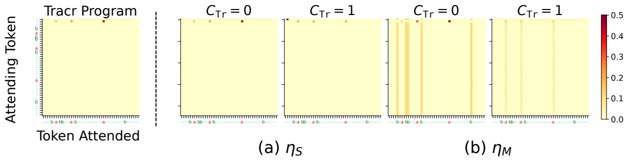


Figure 41: **New capabilities are not learned on using η_S for fine-tuning.** (a) The counting a's capability implemented by the originally compiled Tracr model is to attend to a's. On using η_S (10^{-3}) for fine-tuning, the compiled model is not able to learn the capability of counting b's (b). Increasing the learning rate makes the model learn to attend b's (c), but the pretraining capability of attending to a's still exists.

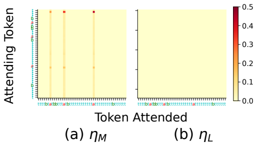


Figure 42: **Capability Revival Analysis:** Using η_{VS} (a) is able to recover the old capability on reverse fine-tuning the model fine-tuned with η_M. But η_S is not able to recover the original capability, when the compiled model is fine-tuned with η_l. This is because using a large value of learning rate during=fine-tuning hampers the pre-training capabilities.

I.2 COUNTER RESULTS

A detailed analysis of the attention maps of block-1 and 2 for different learning rates is shown in Fig. 44. We further validate our results for three different input datapoints in Fig. 40 and Fig. 48. A detailed analysis of the activation map in Block-1 for different values of $C_{\mathtt{Tr}}$ is shown in Fig. 47.

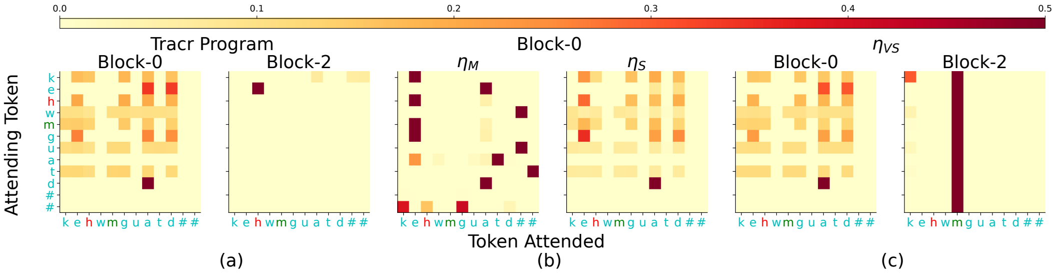


Figure 43: **Learning of the fine-tuning capability is affected by type of compiled capability present in the model.** (a) The Tracr program implements the sorting function in Block-0 and read function in Block-2. Using η_M and η_S can destroy the sorting capability present in Block-1 (b). But using η_{VS}, preserves the sorting capability (c). Thus on using η_{VS}, the model learns to read a different stream of output, while preserving the sorting capability.

Table 3: **Results on counting and max element task Tracr setups.** The evaluation is done on test sets with and without the spurious correlation. The Tracr compiled models are fine-tuned for different learning rates and different value of $C_{\texttt{Tr}}$.

η	$C_{\texttt{Tr}}$	Counting Element				Max Identifier			
		$C_{\texttt{Te}} = 1$		$C_{\texttt{Te}} = 0$		$C_{\texttt{Te}} = 1$		$C_{\texttt{Te}} = 0$	
		Acc. $\texttt{O}_{\texttt{PT}}$	Acc. $\texttt{O}_{\texttt{FT}}$	Acc. $\texttt{O}_{\texttt{PT}}$	Acc. $\texttt{O}_{\texttt{FT}}$	Acc. $\texttt{O}_{\texttt{PT}}$	Acc. $\texttt{O}_{\texttt{FT}}$	Acc. $\texttt{O}_{\texttt{PT}}$	Acc. $\texttt{O}_{\texttt{FT}}$
10^{-1}	0	0.0	100.0	0.0	100.0	20.3	34.5	0.0	99.3
	0.2	0.0	100.0	0.0	100.0	0.5	92.0	0.0	97.1
	0.5	0.0	100.0	0.0	100.0	0.6	97.0	0.1	97.6
	0.6	0.0	100.0	0.0	100.0	0.3	98.5	0.0	96.7
	0.8	0.0	100.0	0.0	100.0	0.1	99.4	0.0	98.6
	0.9	0.0	100.0	0.0	100.0	0.7	98.2	0.1	92.5
	1	0.0	100.0	35.8	0.7	0.3	99.6	16.8	37.8
10^{-2}	0	1.1	96.3	0.0	98.8	29.7	0.2	16.3	52.6
	0.2	0.0	100.0	0.0	99.2	28.4	18.6	19.0	46.0
	0.5	0.6	99.4	0.0	95.9	4.8	87.9	3.3	92.6
	0.6	0.1	99.9	0.0	98.8	3.9	83.2	2.6	82.5
	0.8	0.3	99.6	0.1	97.0	4.5	88.8	6.6	72.9
	0.9	1.4	98.5	7.1	39.3	16.0	45.7	26.9	11.1
10^{-3}	1	0.3	98.3	4.2	0.2	11.1	78.5	23.8	14.4
	0	54.6	1.2	25.7	27.2	6.4	20.2	4.5	28.5
	0.2	50.2	15.0	26.5	24.3	7.4	27.4	5.4	28.0
	0.5	7.1	90.9	19.8	2.3	11.3	24.0	7.6	20.8
	0.6	4.1	94.2	11.8	2.2	11.8	26.7	8.4	20.1
	0.8	1.3	98.3	6.7	0.7	11.5	34.3	8.5	19.9
	0.9	1.8	97.8	9.2	0.7	14.6	32.2	11.4	15.8
10^{-4}	1	4.0	94.3	10.3	2.2	16.0	33.2	12.8	14.0
	0	32.6	0.0	10.6	28.7	0.5	82.6	0.5	91.1
	0.2	59.2	0.1	31.9	24.4	0.1	84.8	0.6	91.3
	0.5	28.5	65.1	37.8	5.6	0.0	89.3	0.6	91.8
	0.6	24.4	70.3	35.6	4.8	0.0	89.6	0.6	90.8
	0.8	14.1	84.2	29.7	2.1	0.0	89.7	0.6	89.9
	0.9	1.3	98.3	6.7	0.7	0.0	93.2	0.2	97.1
	1	1.6	98.3	10.6	0.2	0.0	90.2	0.7	88.6

We present an evidence further, showing that capability of the Tracr compiled model to count $\texttt{a}$'s is still present in the model in Fig. 45, 46, where Fig. 45 presents a closer look of the Fig. 46. As can be seen in Fig. 45, on using η_S and η_M, Block-1 activation map of the Tracr fine-tuned model shows neurons corresponding to token $\texttt{a}$ being activated in a different output channel.

Finally, we present evidence of the wrapper being learned by the model on fine-tuning using spuriously correlated dataset. We show that this wrapper can be localized in a very few neurons of the model.

Table 4: **Results on counting task Tracr for reverse fine-tuning with different learning rates.** Fine-tuning was done using η_M. The evaluation is done on test sets with and without the spurious correlation. The Tracr compiled models are fine-tuned for different learning rates and different value of $C_{\mathtt{Tr}}$.

η	$C_{\mathtt{Tr}}$	Counting Element							
		$C_{\mathtt{Te}} = 1$		$C_{\mathtt{Te}} = 0$		$C_{\mathtt{Te}} = 1$		$C_{\mathtt{Te}} = 0$	
		Acc. $\mathtt{O}_{\mathtt{PT}}$	Acc. $\mathtt{O}_{\mathtt{FT}}$	Acc. $\mathtt{O}_{\mathtt{PT}}$	Acc. $\mathtt{O}_{\mathtt{FT}}$	Acc. $\mathtt{O}_{\mathtt{PT}}$	Acc. $\mathtt{O}_{\mathtt{FT}}$	Acc. $\mathtt{O}_{\mathtt{PT}}$	Acc. $\mathtt{O}_{\mathtt{FT}}$
10^{-1}	0	96.5	100.0	2.1	0.0	94.0	100.0	0.1	0.0
	0.2	98.5	100.0	0.1	0.0	94.9	100.0	0.1	0.0
	0.5	39.4	100.0	43.6	0.0	44.9	100.0	5.6	0.0
	0.6	72.9	100.0	26.7	0.0	69.4	100.0	0.2	0.0
	0.8	49.3	100.0	6.5	0.0	37.1	100.0	16.6	0.0
	0.9	31.3	100.0	64.0	0.0	34.1	100.0	1.7	0.0
	1	69.2	100.0	3.7	0.0	65.4	100.0	6.3	0.0
10^{-2}	0	63.3	99.9	36.6	0.1	65.5	98.6	0.0	0.0
	0.2	19.5	100.0	48.8	0.0	29.6	100.0	17.5	0.0
	0.5	14.3	100.0	54.4	0.0	28.9	99.9	18.7	0.0
	0.6	86.2	99.9	13.8	0.1	78.3	98.6	0.0	0.0
	0.8	65.6	100.0	1.7	0.0	43.7	99.5	10.8	0.0
	0.9	33.3	100.0	27.9	0.0	36.5	100.0	12.6	0.0
	1	99.0	99.6	0.9	0.3	95.2	96.8	0.1	0.1
10^{-3}	0	19.8	99.9	34.9	0.0	23.7	98.6	10.2	0.0
	0.2	3.9	17.1	33.8	78.0	24.4	42.9	14.7	3.0
	0.5	2.0	87.6	33.2	9.5	18.9	85.7	11.9	0.9
	0.6	7.1	99.8	35.4	0.1	22.3	97.3	15.4	0.1
	0.8	11.6	97.2	45.8	0.3	27.3	95.5	16.5	0.5
	0.9	24.2	75.2	20.3	23.6	33.5	68.8	13.1	1.5
	1	68.5	99.9	26.7	0.1	65.3	98.6	5.8	0.0
10^{-4}	0	45.2	99.9	0.1	0.1	16.9	98.5	4.9	0.0
	0.2	30.1	9.4	22.7	45.1	18.1	26.2	17.9	19.5
	0.5	30.1	3.2	22.7	41.3	15.7	26.1	17.9	16.2
	0.6	54.1	0.0	45.9	35.9	0.0	27.8	25.7	11.9
	0.8	26.8	83.0	64.5	10.9	3.0	76.8	49.2	1.3
	0.9	27.9	85.0	61.5	8.4	2.1	80.9	44.2	1.1
	1	45.2	99.6	31.9	0.3	42.2	96.8	12.6	0.0

As shown in Fig. 54, we present this evidence for different values of $C_{\mathtt{Tr}}$ in Fig. 55. Similar to the analysis presented for PCFG where we prune multiple neurons, we analyze the effect of pruning of mutliple weights and neurons in Fig. 56 and Fig. 57 respectively. These results verify that the wrapper learned by the Tracr compiled model on fine-tuning using spuriously correlated dataset can indeed be localized to a few weights of the Tracr model. To ensure that the gains achieved on pruning are indeed because of removal of the wrapper, we present the histograms showing the distribution of the predicted classes in Fig. 58, where it can be observed that after pruning the model still predicts multiple classes.

I.3 Max identifier results

In this section, we provide additional evidence and a detailed analysis of the performance of the Tracr compiled model on the max identifier task. We show that the model implements the sorting pattern in the activation map of its first block in Fig. 52 and Fig-53. We present validation of our observations on considering the spurious correlation as the difference between the fifth and seventh maximmum element being three in Fig. 50. We validate our results for three different input data-points in Fig. 49. A detailed visualization of the attention maps in Block-0 and Block-2 for different learning rates is shown in Fig. 51.

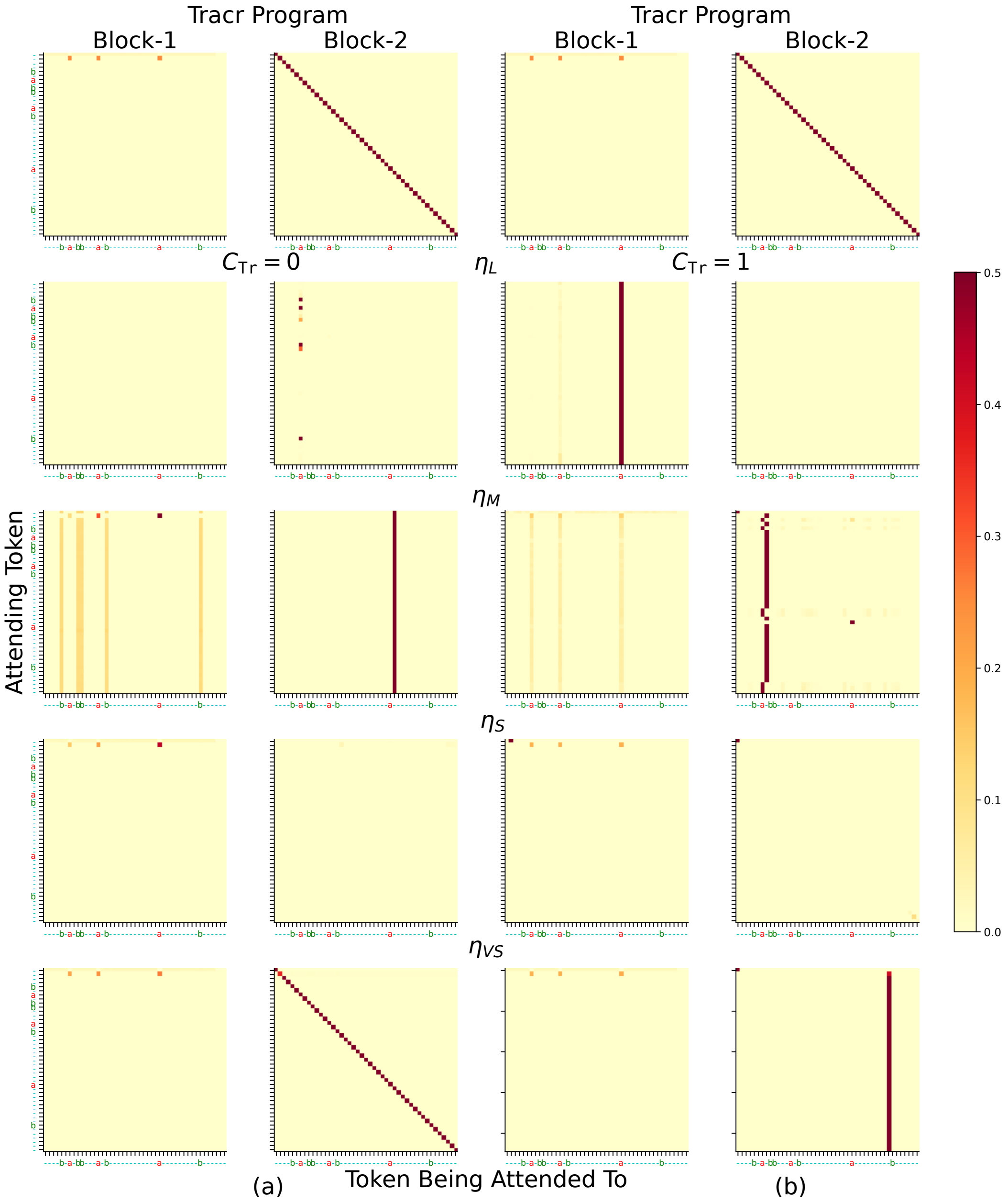


Figure 44: **Counter Task: Visualization of the attention maps of the first and second blocks of the Tracr fine-tuned models.** (a) shows the analysis when the spurious correlation is not present in the fine-tuning datatset, whereas in case of (b) the spurious correlation is present in the fine-tuning dataset. The first row shows the maps for the Tracr compiled model and other rows shows the analysis for different learning rates. **Observation:** (a) Using η_S or η_{VS} the model is not able to learn to attend b's and thus the fine-tuning task performance is poor. Whereas using η_M the model is able to learn to attend to b's, however the capability to count a's is likely still present since the model still attends to a's. Further increasing the learning rate leads to distortion of the compiled capabilities, and thus model learns the fine-tuning task by learning a different capability. (b) In the presence of spurious correlation, even for large learning rate the compiled capability is still present, since the model attends to a's.

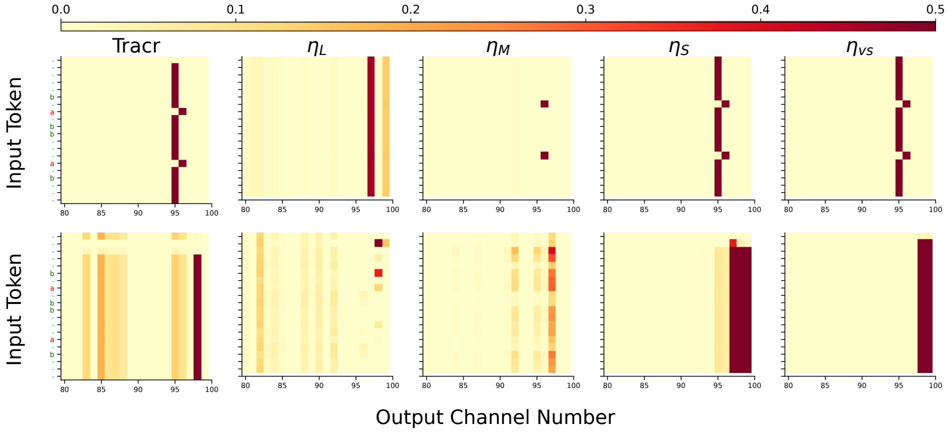


Figure 45: **Counter Task: Visualization of the activated output of the first MLP layer in first and second blocks.** The visualization is shown only for channel numbers 80-100. **Observation:** Using η_M preserves the Tracr compiled capability, while also learning the fine-tuning task. This shows that the model changes behaviourally but mechanistically the compiled capabilities are still present in it.

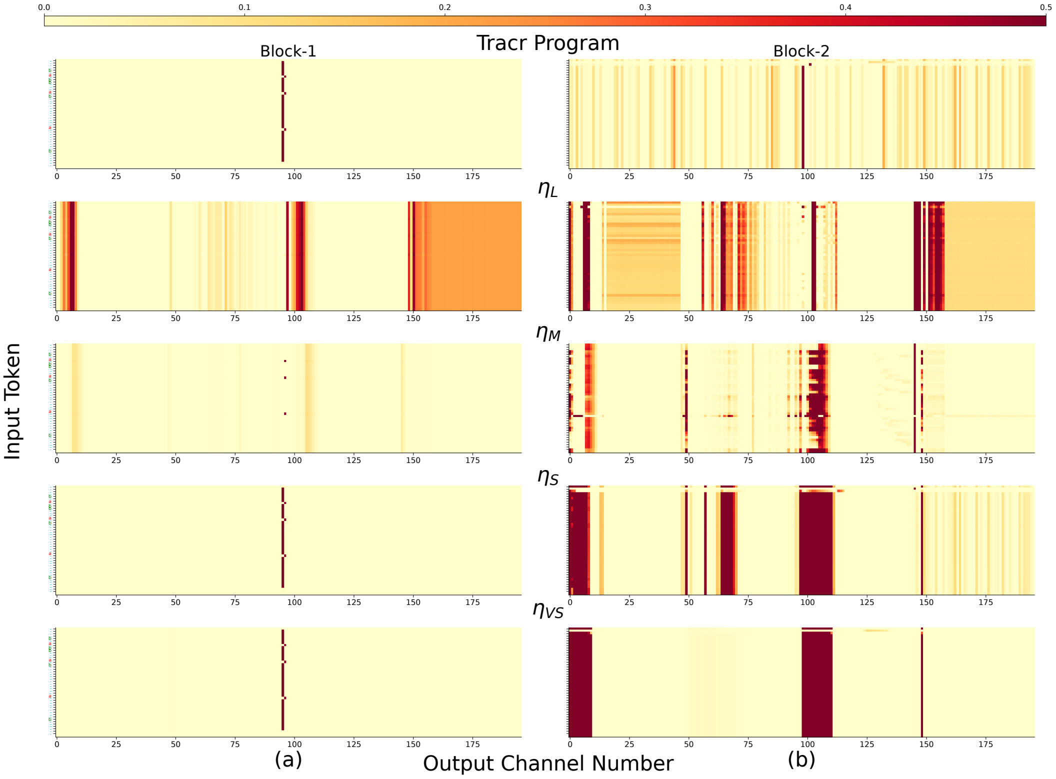


Figure 46: **Counter Task: Visualization of the activated output of the first MLP layer in first and second blocks.** This is the complete visualization of the activation map presented in Fig. 45.

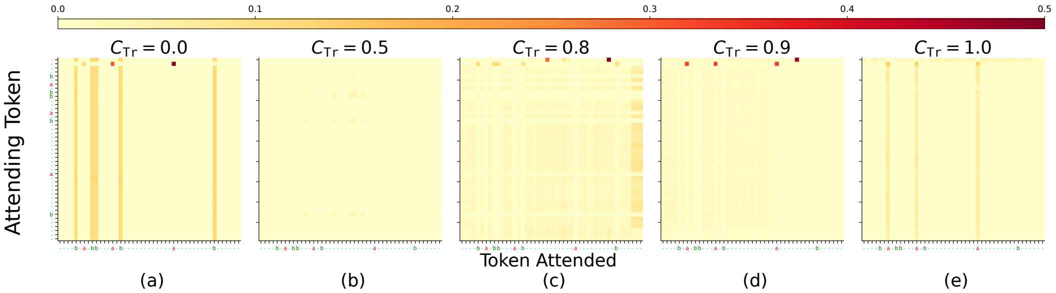


Figure 47: **Counter Task: Effect of the presence of the spuriously correlated data-points in different fractions ($C_{\mathtt{Tr}}$) in the fine-tuning dataset.** The Tracr compiled model with the capability to count a's is fine-tuned to count b's on different values of $C_{\mathtt{Tr}}$. η_M is used for fine-tuning. **Observation:** On increasing the value of $C_{\mathtt{Tr}}$, the model gives lower attention to b's and in case of $C_{\mathtt{Tr}} = 1$, almost no attention is given by the model to b's.

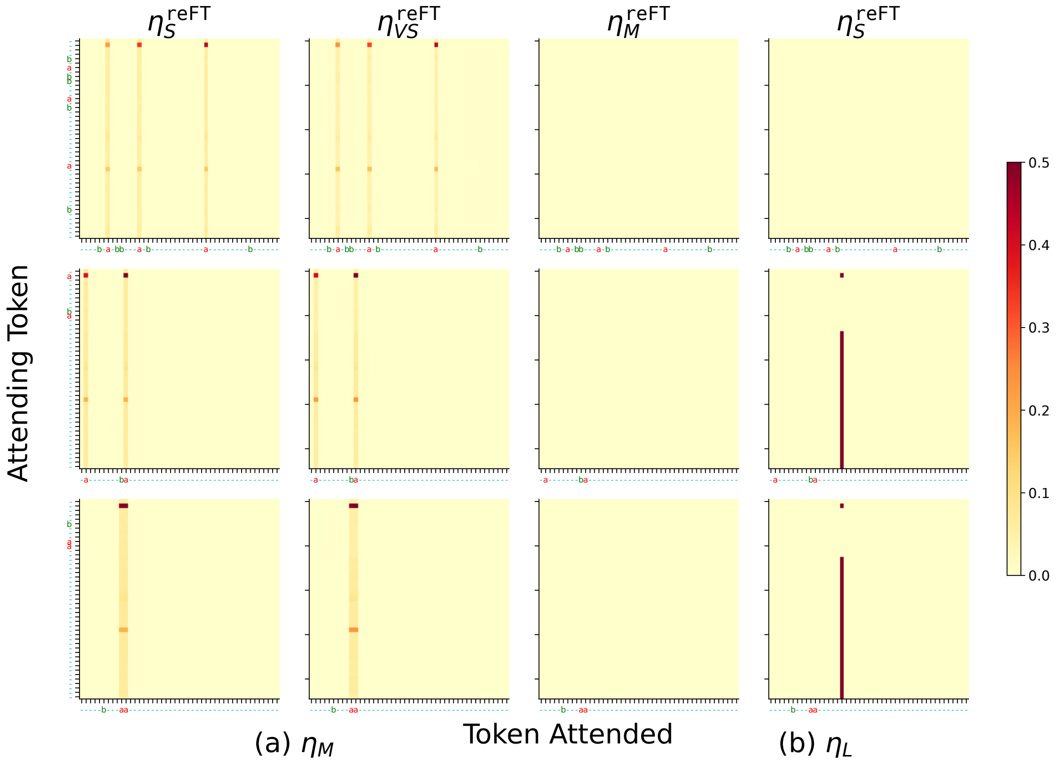


Figure 48: **Counter Task: Validation of Tracr observations on Counter task on reverse fine-tuning on three different input samples.** The rows represents different input samples. **Observation:** Capability revival is possible on using η_M for fine-tuning but not on using η_L.

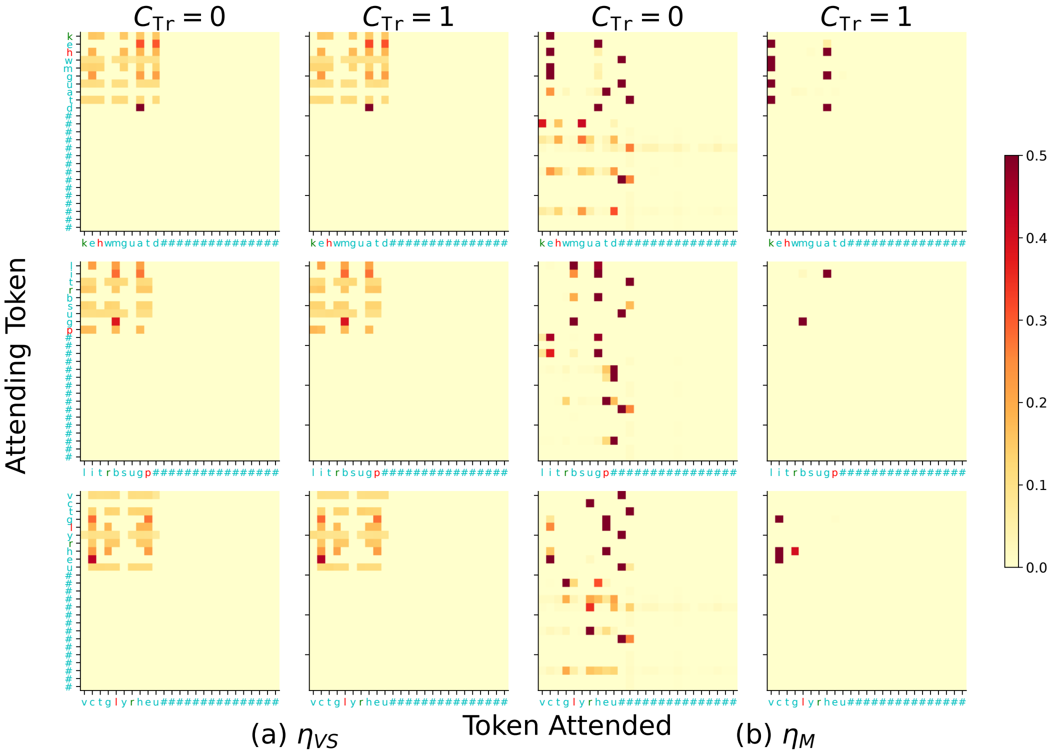


Figure 49: **Max Identifier Task: Validation of Tracr observations on max identifier task on three different input samples.** The rows represents different input samples. **Observation:** Using η_{VS} preserves the original Tracr capabilities and therefore performs well on the fine-tuning task, whereas using η_M distorts the compiled capabilities resulting in poor performance on the fine-tuning task.

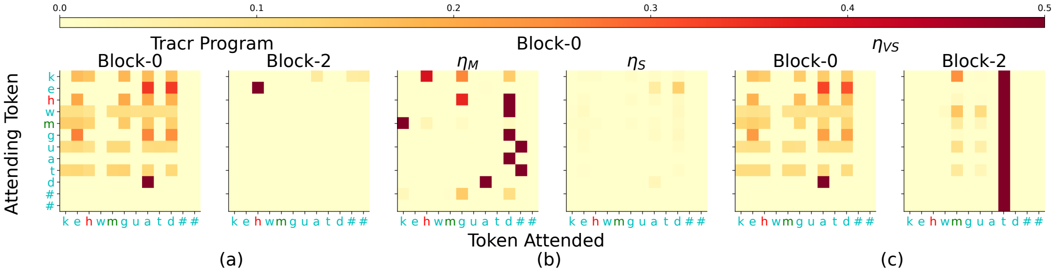


Figure 50: **Max Identifier Task: Validation of Tracr observations on max identifier task with the spurious correlation defined as the difference between the indices of fifth and seventh maximum elements being three.** The rows represents different input samples. **Observation:** Using η_{VS} preserves the original Tracr capabilities and therefore performs well on the fine-tuning task, whereas using η_M distorts the compiled capabilities resulting in poor performance on the fine-tuning task.

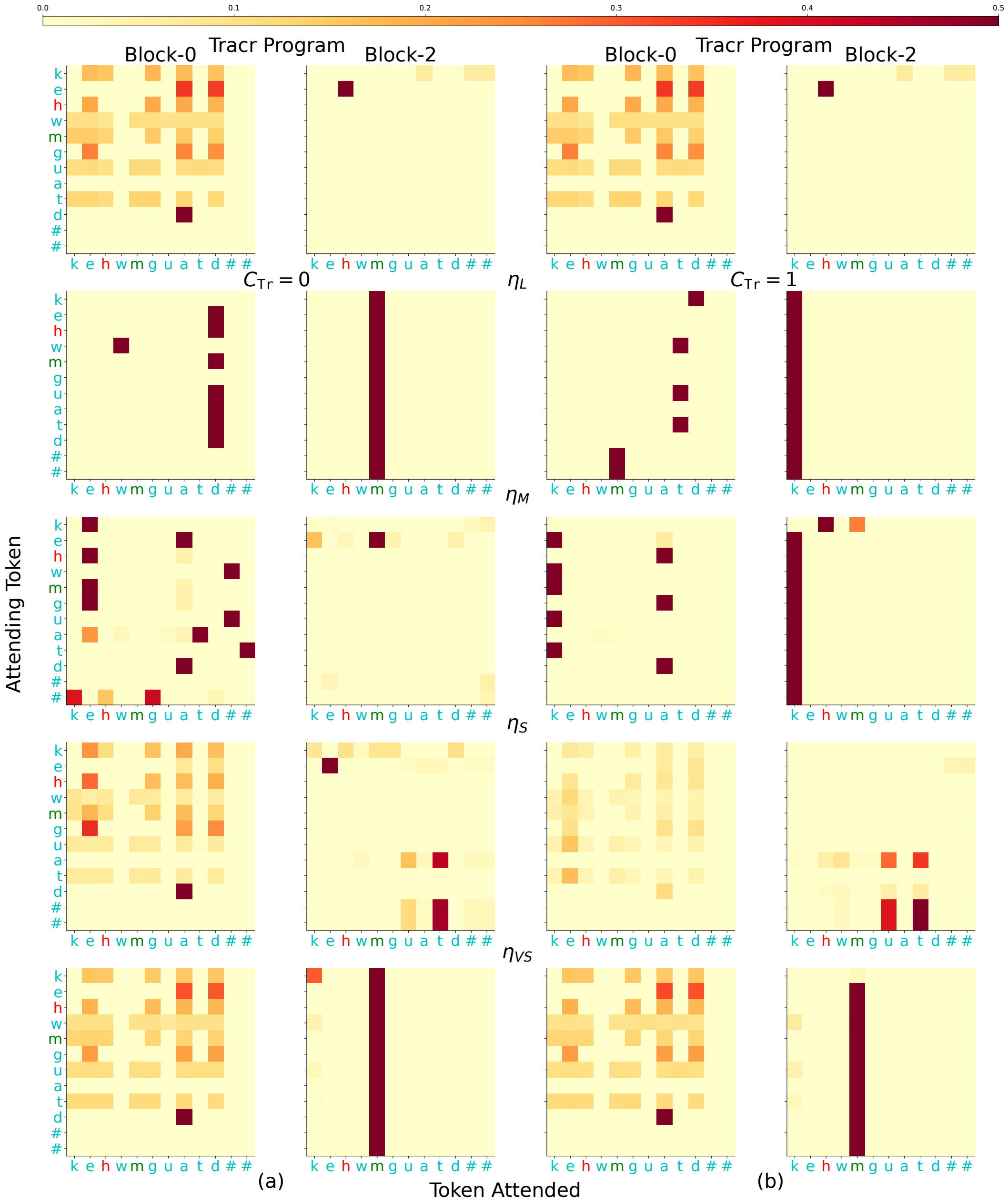


Figure 51: **Max Identifier Task: Visualization of the attention maps of the zeroth and second blocks of the Tracr fine-tuned models on the max identifier task.** (a) shows the analysis when the spurious correlation is not present in the fine-tuning datatset, whereas in case of (b) the spurious correlation is present in the fine-tuning dataset. The first row shows the maps for the Tracr compiled model and other rows shows the analysis for different learning rates. **Observation:** Using η_L, η_M or η_S for fine-tuning distorts the capability of the programmed Tracr model in the Block-0 and as a result the Block-2 attention map is not able to attend to the desired output token. Whereas using η_{VS} is able to preserve the capability and as a result the fine-tuned model is able to attend to the correct token in the attention map in Block-2.

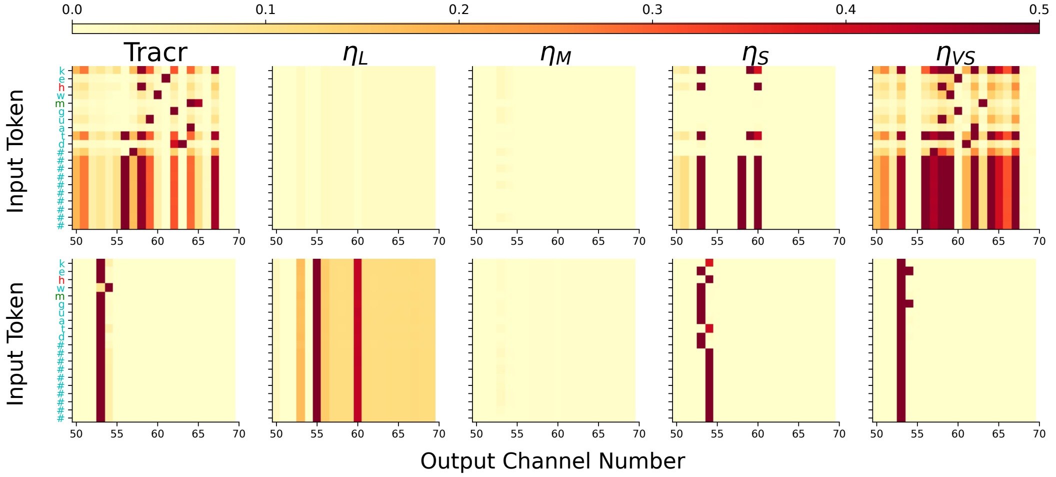


Figure 52: **Max Identifier Task: Visualization of the activated output of the first MLP layer in first and second blocks for the max identifier task.** The visualization is shown only for channel numbers 50-70. **Observation:** Using η_{VS} for fine-tuning, which enables the model to learn the fine-tuning task, preserves the Tracr compiled model's compiled capability of sorting tokens in Block-1. Whereas other learning rates are not able to preserve this capability.

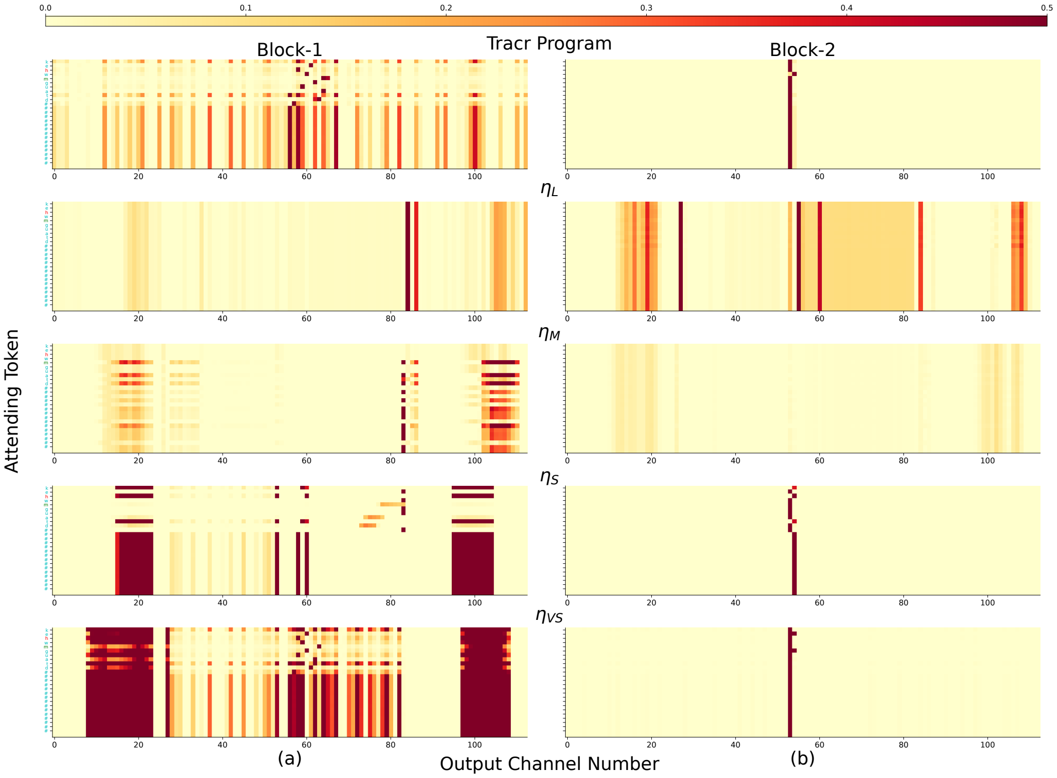


Figure 53: **Max Identifier Task: Visualization of the activated output of the first MLP layer in first and second blocks.** This is the complete visualization of the activation map presented in Fig. 52.

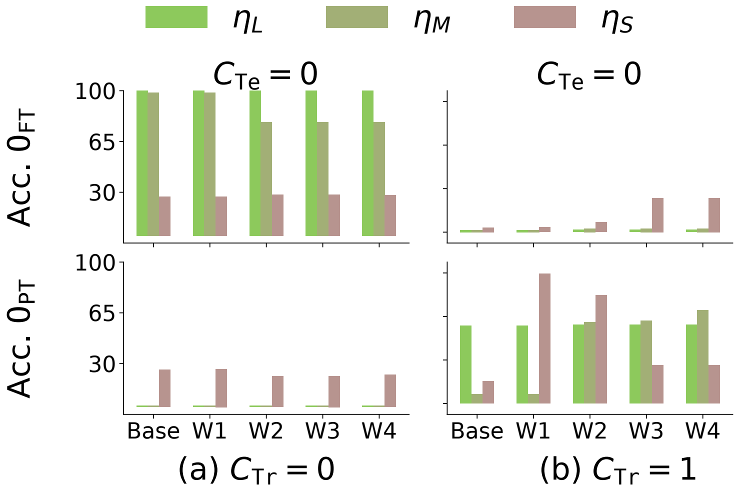


Figure 54: **Counter Task: Pruning evaluation on Tracr model fine-tuned to count `b`'s. Observation:** Higher value of $C_{\mathtt{Tr}}$ leads to the learning of the wrapper on top of the Tracr compiled capability. This wrapper is learned on using η_M and η_S and can be localized in a few weights of the model.

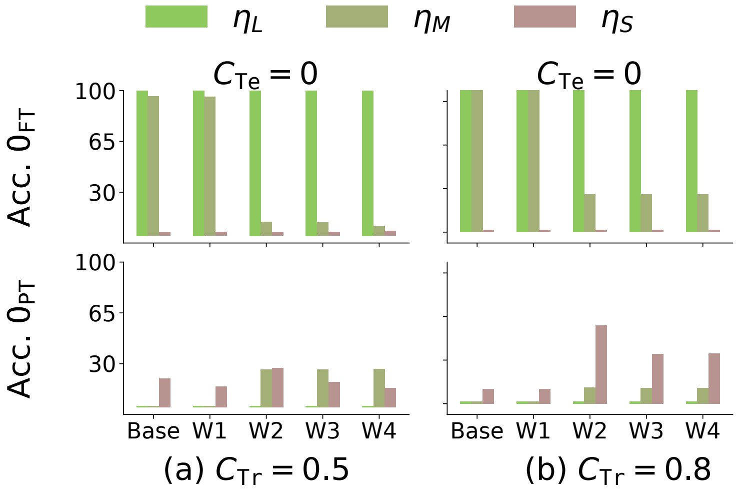


Figure 55: **Counter Task: Pruning evaluation on Tracr model fine-tuned to count `b`'s. Observation:** Observations are consistent with Fig-3.

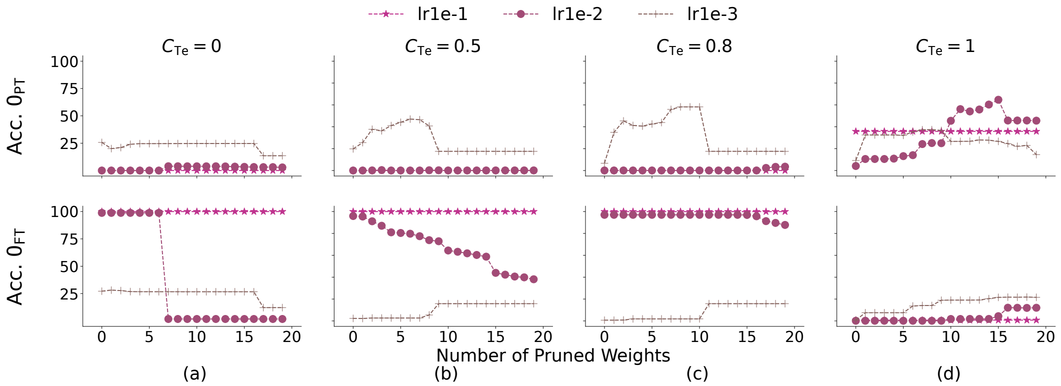


Figure 56: **Counter Task, weight pruning: Pruning weights of the Tracr model fine-tuned to count b's using different learning rates. Observation:** In the presence of spurious correlation the model learns a wrapper when learning rates η_M and η_S are used. This wrapper can be localized in a few weights of the model.

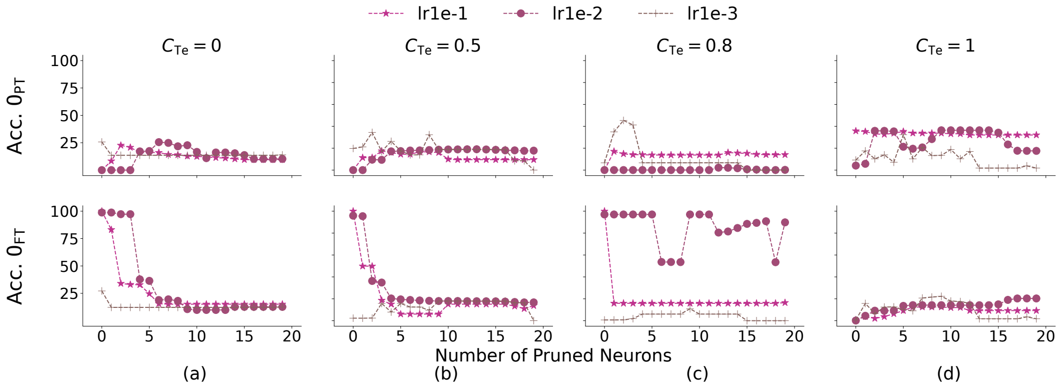


Figure 57: **Counter Task, neuron pruning: Pruning neurons of the Tracr model fine-tuned to count b's using different learning rates. Observation:** In the presence of spurious correlation the model learns a wrapper in case of η_M and η_S. This wrapper can be localized in a few weights of the model.

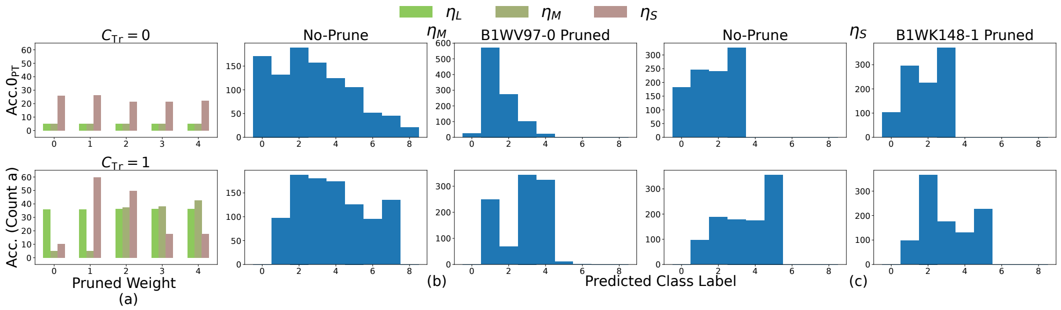


Figure 58: **Counter Task: Effect of pruning a single weight on the distribution of predicted class labels. Observation:** Even after pruning, the model predicts different classes indicating that the gain in accuracy on pruning is indeed because the model has removed the wrapper.

J Additional PCFG results

In this section, we provide a detailed analysis of the PCFG results on the counter task. More specifically, we analyze the effect of the presence of weakly and strongly relevant capability in the model across three different parameters: training iterations (n_{iters}), fraction of fine-tuning data samples with spurious correlation ($C_{\texttt{Tr}}$) and the probability of sampling operand token (`O`) to be `a` during pre-training ($\mathcal{P}_{\texttt{T}}$). $\mathcal{P}_{\texttt{T}}$ essentially determines whether the capability present in the pre-trained model is strongly relevant or weakly relevant for the fine-tuning task. Additionally we also analyze the effect of using the spuriously correlated ($C_{\texttt{Te}} = 1$) and uncorrelated test set ($C_{\texttt{Te}} = 0$) for evaluation of the fine-tuned model. We present the summary of the results in Tables 5, 6 for the counting task and Tables 7,6 for the index of occurrence tasks. Then we discuss the effect of learning rate on fine-tuning pre-trained models with weakly and strongly relevant capabilities in Fig. 59, 60. We observe that the observations are consistent for the considered counting and index of occurrence tasks. Next we analyze the effect of the presence of weakly and strongly relevant capability in the pre-trained model for different fractions of spuriously correlated data-points ($C_{\texttt{Tr}}$) and different pre-training iterations (n_{iters}) in Fig. 61, 63, 65 for the counting element task and Fig. 62, 64, 66 for the index of occurrence task. We observe that the observations are fairly consistent across both the tasks and different values of n_{iters}. Next we present the effect of the spuriously correlated data and presence of weakly and strongly correlated capabilities on the learning of the wrapper in Fig. 67, 68 on using uncorrelated test set for evaluation on counting and index of occurrence tasks respectively. Similar analysis on using test set with spuriously correlated samples is present in Fig. 69 and 70. We present the capability revival analysis on the Counter task for $n_{iters} = 200K$ and $n_{iters} = 50K$ pre-trained models for weakly and strongly relevant capability fine-tuned models in Fig. 75 and Fig. 76 respectively. A similar analysis for different number of pre-training iterations is present in Fig. 77.

Table 5: Results on the PCFG counting task with 200K iterations of pre-training.

η	$\mathcal{P}_{\texttt{T}}$ (`a`)	$C_{\texttt{Tr}}$	Acc $\texttt{O}_{\texttt{PT}}$					Acc $\texttt{O}_{\texttt{FT}}$				
			Acc PT	$C_{\texttt{Te}} = 0$		$C_{\texttt{Te}} = 1$		Acc PT	$C_{\texttt{Te}} = 0$		$C_{\texttt{Te}} = 1$	
				$1K$	$10K$	$1K$	$10K$		$1K$	$10K$	$1K$	$10K$
10^{-4}	0.999	0	100	9.7	9.5	14.5	0	27.1	99.4	100	74.9	87.5
		0.5		7.2	9.7	1.3	0		75.9	99.9	95.9	100
		0.8		5.6	10.8	0.2	0		60	99.8	98.1	100
		1		17.2	15.2	0.2	0		0	1.6	98.9	100
	0.9	0	100	99.9	92.6	13.6	67.2	99.9	99.8	100	100	92.8
		0.5		100	90.7	15.6	73.5		99.8	99.4	99.9	100
		0.8		99.9	43.4	11.4	33.2		99.4	99.2	100	100
		1		99.8	15.9	16.5	2.4		99.6	9.4	6.1	100
	0.5	0	99.9	98.9	14.7	76.6	44.1	99.9	99.9	100	87.6	99.2
		0.5		95.1	23.6	71.2	33.4		99.7	100	99.8	99.9
		0.8		92.8	12.2	79.8	2.2		99.9	99.9	98.7	99.9
		1		49.5	19	60.6	0		25.8	16	99.8	100
10^{-5}	0.999	0	100	48.9	10.2	51.9	4.6	27.1	39.8	99.8	13.9	79.7
		0.5		19.7	11.6	12.3	1.2		18.4	98.1	81.4	99.7
		0.8		12.1	6.6	7.7	0.2		6.1	85.7	98.4	99.7
		1		0.4	17.5	0	0		0	0	99.9	100
	0.9	0	100	100	85.3	94.8	56.9	99.9	99.8	99.9	83.3	87.3
		0.5		99.9	67.2	94.9	55.4		99.9	99.9	99.3	99.8
		0.8		100	34.6	94.8	21.7		99.8	99.4	99.7	100
		1		98.5	13.5	88.6	0.8		58.3	3.6	99.8	100
	0.5	0	100	100	97.5	97.5	65.7	99.9	100	100	95.6	95.4
		0.5		99.9	94.1	98.1	69		100	100	99.3	100
		0.8		99.9	87.4	93.8	67.7		99.8	100	100	100
		1		99.6	41.2	91.8	53.3		90.1	19.6	99.8	100
10^{-6}	0.999	0	100	100	29	96.6	25.7	27.1	28.5	51.8	15.1	29.2
		0.5		98.7	21.8	88.9	10.6		23.3	23.7	20.3	87.5
		0.8		83	15.1	69.7	6.8		18.4	8.9	26.5	99.7
		1		71.7	2.3	56	0		15.7	0	29.5	99.9
	0.9	0	100	100	100	95.4	91.8	99.9	99.8	99.5	84.1	84.6
		0.5		100	99.9	96	94.4		99.6	99.5	95.9	99.6
		0.8		99.8	99.4	95.9	92.6		99.6	99.3	94.8	99.6
		1		99.8	51.6	95.1	63.9		99.5	30.5	94.2	99.7
	0.5	0	99.9	100	99.9	97.7	98.1	99.9	99.8	99.8	95.4	95.6
		0.5		100	99.9	97.9	93.7		100	99.9	98.5	99.6
		0.8		100	100	98.8	93.1		100	99.9	98.3	99.9
		1		100	97.6	98.6	85.1		99.8	73.9	98.3	100

Table 6: Results on the PCFG counting task with 50K iterations of pre-training.

η	$\mathcal{P}_{\mathtt{T}}$ (a)	$C_{\mathtt{Tr}}$	Acc $\mathtt{O}_{\mathtt{PT}}$					Acc. $\mathtt{O}_{\mathtt{FT}}$				
			Acc. PT	$C_{\mathtt{Te}} = 0$		$C_{\mathtt{Te}} = 1$		Acc. PT	$C_{\mathtt{Te}} = 0$		$C_{\mathtt{Te}} = 1$	
				$1K$	$10K$	$1K$	$10K$		$1K$	$10K$	$1K$	$10K$
10^{-4}	0.999	0	99.9	10.8	9.1	2	0.1	5.17	98.9	100	86.3	93.6
		0.5		11.7	8.9	2.1	0.1		90.2	99.9	97.9	99.8
		0.8		5.5	11	0	0		64.9	100	98.9	99.9
		1		20.2	15.9	0	0		0	1.9	99.9	100
	0.9	0	99.9	9.1	10.3	0.7	0.1	15.8	99.6	100	84.2	94.4
		0.5		11.4	10.6	1.5	0		93.2	99.9	97.9	100
		0.8		4.2	9.2	0.1	0		63.1	100	97.8	100
		1		18.4	16.4	0	0		0.5	5	100	99.9
	0.5	0	99.8	87.7	10.1	62.6	0	99.7	99.9	100	89.3	93.5
		0.5		90.1	9.4	67.4	0		99.5	100	99.9	100
		0.8		59.5	10.1	29.3	0.1		99.2	99.9	100	99.9
		1		18.7	15.2	5.1	0		17.4	14.2	100	100
10^{-5}	0.999	0	99.9	3	9.3	16.6	0.6	5.17	32.6	99.8	12.4	88.7
		0.5		30.9	11.4	4.1	0.6		12.7	99.1	93	99.7
		0.8		6.3	10.8	1	0.1		1.4	93.9	99	99.8
		1		2.1	20.7	0.2	0		0	0	99.8	99.9
	0.9	0	99.9	28	10.9	34.7	0.1	15.8	39.6	99.8	23.4	88.6
		0.5		33.2	8.9	4.2	0		22.7	99.6	92.8	100
		0.8		13.1	9.3	2.9	0		9.4	95.1	98.9	100
		1		1.9	19.8	0.3	0		0	0.1	99.7	100
	0.5	0	99.8	99.6	73.7	88.1	46	99.7	99.9	99.9	86.4	89.2
		0.5		99.6	79.3	82.7	57.1		99.8	99.9	99.1	99.9
		0.8		99.6	60.8	80	33.6		99.5	99.9	99.3	100
		1		81.7	12.9	68.6	0.5		46.4	16.1	98.8	100
10^{-6}	0.999	0	99.9	94.1	18.6	81.9	18.8	5.17	9	49.1	0.4	24.4
		0.5		38.8	37.2	20	5.6		6	23.8	50.7	93.7
		0.8		14.4	8.8	4.2	0.2		0.9	7.8	74.8	99.9
		1		8.9	5.8	2.3	0.2		1.3	0	79.2	99.8
	0.9	0	99.9	99.7	21	94.3	20.6	15.8	24.4	56.8	14.6	28.4
		0.5		46.9	38.6	19.7	3.6		11.8	28.6	39.4	92
		0.8		30.4	10.4	8.7	1.6		3.6	12.3	62.8	98.9
		1		27	6.3	6.7	0.4		1.3	0	70.4	99.5
	0.5	0	99.8	99.8	99.6	94.4	82.7	99.7	99.9	99.8	81	84.6
		0.5		100	99.5	91.4	80.1		99.7	99.9	95.9	99.6
		0.8		99.7	97	90.1	74.6		99.7	99.3	96.2	99.3
		1		99.8	55.7	91.2	61		99.4	30	96.3	99.4

Table 7: Results on the PCFG index of occurrence task with 200K< iterations of pre-training.

η	$\mathcal{P}_{\texttt{T}}$ (a)	$C_{\texttt{Tr}}$	Acc. $\texttt{O}_{\texttt{PT}}$					Acc $\texttt{O}_{\texttt{FT}}$				
			Acc. PT	$C_{\texttt{Te}} = 0$		$C_{\texttt{Te}} = 1$		Acc. PT	$C_{\texttt{Te}} = 0$		$C_{\texttt{Te}} = 1$	
				$1K$	$10K$	$1K$	$10K$		$1K$	$10K$	$1K$	$10K$
10^{-4}	0.999	0	99.0	5.8	0.0	23.9	0.0	9.3	71.8	99.7	46.1	99.5
		0.5		9.0	0.0	23.0	0.0		66.6	99.6	81.0	100.0
		0.8		14.9	0.0	0.8	0.0		36.8	99.1	99.5	100.0
		1		33.6	9.5	0.0	0.0		3.7	5.8	100.0	100.0
	0.9	0	99.2	96.2	0.0	88.6	0.1	97.1	98.5	99.9	98.6	99.7
		0.5		96.8	0.0	83.5	0.0		97.3	99.5	100.0	100.0
		0.8		96.8	0.0	84.8	0.0		97.4	99.2	100.0	100.0
		1		72.6	1.3	79.1	0.0		37.3	24.0	100.0	100.0
	0.5	0	98.0	95.7	4.5	96.5	3.6	98.9	99.5	99.7	99.8	99.8
		0.5		95.2	5.6	78.2	3.5		99.1	99.8	100.0	100.0
		0.8		96.0	17.7	90.2	5.6		98.8	99.4	100.0	100.0
		1		91.3	15.8	79.8	14.5		48.0	28.8	100.0	100.0
10^{-5}	0.999	0	99.0	94.6	2.6	86.2	18.6	9.3	17.0	85.4	16.0	68.5
		0.5		98.4	3.4	97.6	7.7		14.3	79.8	27.2	97.8
		0.8		94.1	5.1	94.0	0.2		10.7	66.6	37.6	99.7
		1		77.6	27.7	70.8	0.0		4.9	3.7	56.6	100.0
	0.9	0	99.2	99.2	88.5	99.0	79.3	97.1	97.3	99.1	98.9	99.1
		0.5		99.2	91.9	99.2	77.9		97.3	98.3	100.0	100.0
		0.8		98.8	95.6	98.9	84.3		96.9	98.7	99.9	100.0
		1		98.0	65.4	98.0	72.7		81.3	31.8	99.9	100.0
	0.5	0	98.0	97.3	95.8	100.0	89.1	98.9	99.1	99.9	99.9	99.4
		0.5		97.6	95.9	98.8	75.4		99.2	99.3	100.0	100.0
		0.8		97.4	95.1	97.8	80.3		99.6	99.1	100.0	100.0
		1		97.4	87.7	98.0	81.0		97.2	51.8	100.0	100.0
10^{-6}	0.999	0	99.0	99.0	80.4	99.4	73.0	9.3	14.6	23.3	19.5	15.5
		0.5		98.9	94.2	99.5	94.3		13.7	20.2	22.0	40.5
		0.8		99.2	67.8	98.7	61.2		13.2	9.2	25.1	71.8
		1		99.4	61.6	98.8	11.3		12.9	4.5	19.3	95.6
	0.9	0	99.2	98.8	99.6	99.5	99.6	97.1	97.3	97.0	98.0	98.4
		0.5		99.0	98.8	99.8	99.2		96.6	96.6	98.8	99.9
		0.8		98.7	99.2	99.7	99.0		97.1	96.7	99.3	100.0
		1		98.7	98.0	99.4	95.8		97.8	70.7	99.3	100.0
	0.5	0	98.0	97.8	97.9	99.8	99.6	98.9	99.2	98.9	99.2	99.7
		0.5		98.7	98.0	99.8	98.2		98.9	99.0	99.8	100.0
		0.8		97.4	97.9	99.6	98.2		99.5	98.9	100.0	100.0
		1		98.1	96.7	99.9	96.4		98.9	94.5	99.9	100.0

Table 8: Results on the PCFG index of occurrence task with 50K iterations of pre-training.

η	$\mathcal{P}_{\texttt{T}}\,(\texttt{a})$	$C_{\texttt{Tr}}$	Acc $\texttt{O}_{\texttt{PT}}$					Acc $\texttt{O}_{\texttt{FT}}$				
			Acc. PT	$C_{\texttt{Te}} = 0$		$C_{\texttt{Te}} = 1$		Acc. PT	$C_{\texttt{Te}} = 0$		$C_{\texttt{Te}} = 1$	
				$1K$	$10K$	$1K$	$10K$		$1K$	$10K$	$1K$	$10K$
10^{-4}	0.999	0	94.2	0.5	0.0	11.7	0.1	3.2	77.1	99.6	66.1	99.4
		0.5		3.2	0.0	2.0	0.0		60.0	98.5	92.5	100.0
		0.8		6.7	0.1	1.0	0.0		26.7	96.8	98.7	100.0
		1		23.1	10.9	0.0	0.0		3.8	5.1	99.6	100.0
	0.9	0	94.2	43.2	0.0	34.6	0.8	69.9	86.1	99.6	75.1	97.5
		0.5		48.2	0.0	56.2	0.1		81.0	98.9	97.5	99.9
		0.8		53.5	0.0	53.1	0.0		74.0	97.0	98.8	100.0
		1		12.8	11.0	32.8	0.0		4.4	3.4	99.8	100.0
	0.5	0	88.6	72.1	2.3	59.6	2.6	91.5	95.2	98.6	90.2	99.6
		0.5		65.4	2.3	70.4	0.0		91.4	99.3	98.8	100.0
		0.8		56.6	1.5	65.5	0.0		88.7	97.4	99.7	100.0
		1		39.0	12.8	40.2	0.0		6.1	5.3	99.9	100.0
10^{-5}	0.999	0	94.2	5.0	0.1	6.4	7.7	3.2	18.4	88.9	10.3	86.5
		0.5		74.1	1.4	42.3	0.4		10.1	77.0	55.2	98.8
		0.8		37.4	3.8	13.2	0.7		6.9	45.9	86.9	98.7
		1		48.0	19.1	2.7	0.0		0.5	4.4	95.2	100.0
	0.9	0	94.2	89.2	18.9	69.6	26.8	69.9	74.0	91.1	66.8	82.8
		0.5		91.9	25.0	75.9	38.9		70.8	86.9	82.0	99.2
		0.8		92.4	30.4	75.9	32.1		60.8	80.2	87.8	99.8
		1		69.7	8.8	73.9	8.4		13.8	4.4	92.6	100.0
	0.5	0	88.6	87.6	64.0	77.7	51.4	91.5	93.1	95.6	85.5	90.9
		0.5		85.9	57.5	80.0	57.0		90.4	95.5	93.9	100.0
		0.8		84.9	46.8	78.2	50.4		89.2	91.6	96.8	100.0
		1		80.1	33.5	77.6	13.3		38.6	5.5	98.1	100.0
10^{-6}	0.999	0	94.2	87.5	3.7	73.8	14.2	3.2	8.3	23.0	15.8	20.7
		0.5		95.3	49.3	83.3	19.2		5.4	13.7	27.5	76.9
		0.8		94.0	31.0	71.4	7.7		3.9	7.2	38.7	93.7
		1		88.1	43.0	50.6	2.1		0.3	1.1	44.7	97.5
	0.9	0	94.2	93.6	88.2	73.2	70.7	69.9	69.6	75.6	56.9	66.9
		0.5		94.3	92.2	73.1	76.4		70.3	69.7	67.4	88.6
		0.8		92.7	89.8	75.7	80.3		65.3	63.6	71.8	94.7
		1		95.6	56.2	72.8	72.3		60.2	7.8	71.9	97.9
	0.5	0	88.6	87.3	87.0	80.0	76.6	91.5	90.3	93.8	85.1	88.3
		0.5		87.7	84.5	80.4	80.8		93.4	91.5	90.3	97.6
		0.8		90.6	83.2	83.1	80.5		92.3	87.9	90.1	98.6
		1		89.0	77.3	79.8	70.2		92.4	23.0	91.9	99.7

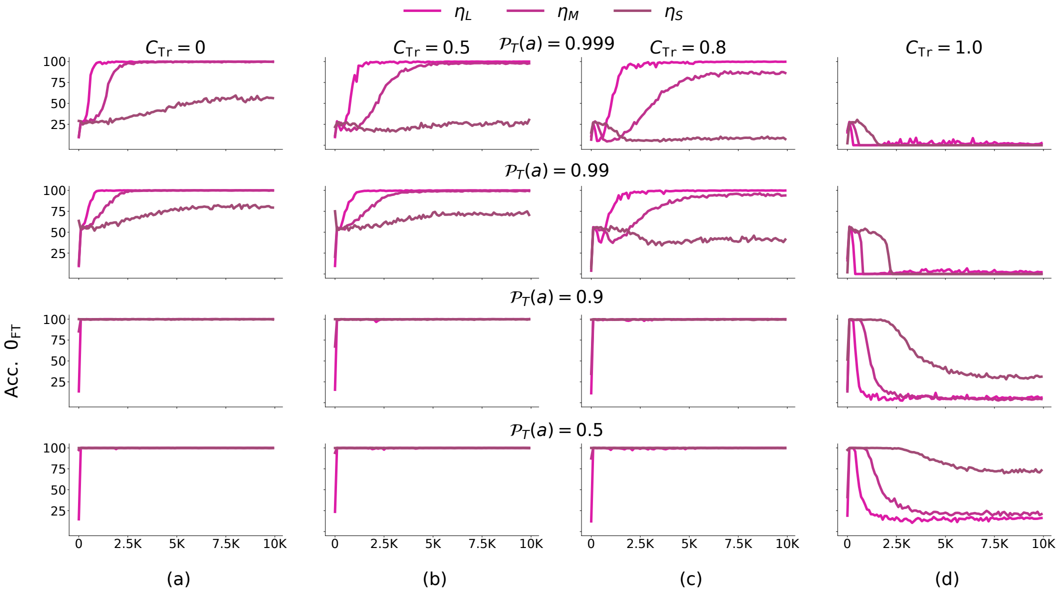


Figure 59: **Counter Task, $n_{iters} = 200K$, $C_{\texttt{Te}} = 0$:** Effect of learning rate (LR) on fine-tuning pre-trained models with weakly and strongly relevant capabilities and using different values of $C_{\texttt{Tr}}$ for fine-tuning. **Observation:** In the presence of strongly relevant capability, training with η_S yields good performance on the fine-tuning dataset. The convergence time to learn the fine-tuning task increases with an increase in $C_{\texttt{Tr}}$.

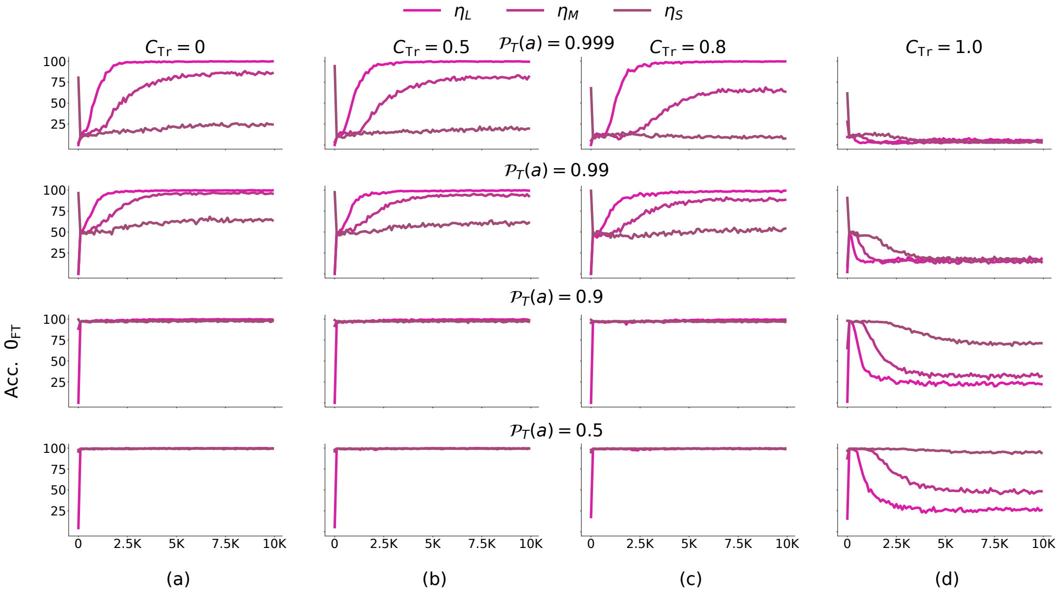


Figure 60: **Index of Occurrence Task, $n_{iters} = 200K$, $C_{\texttt{Te}} = 0$.** The settings are consistent with Fig. 59

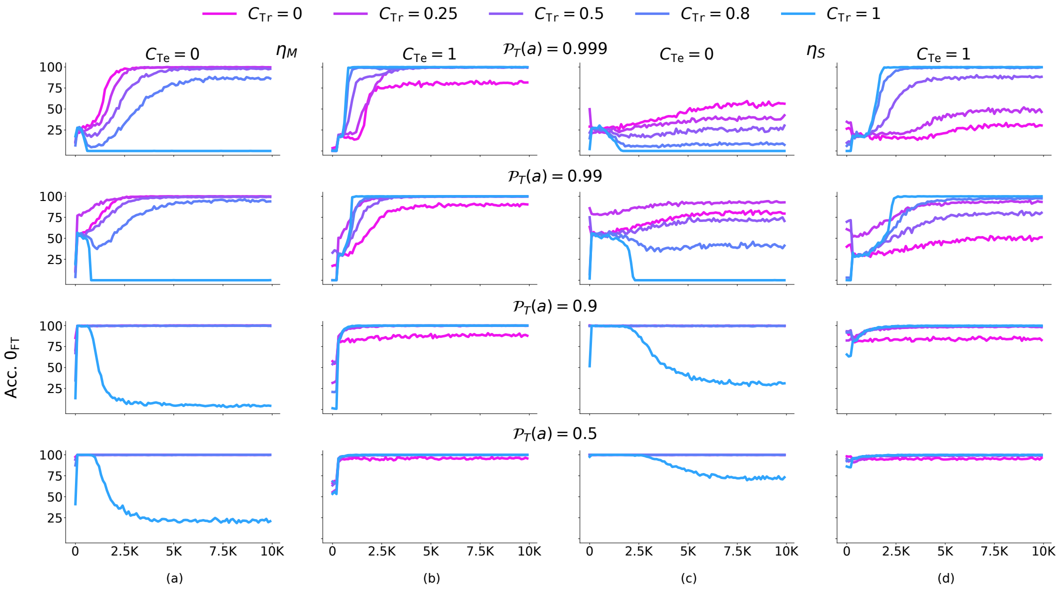


Figure 61: **Counter Task,** $n_{iters} = 200K$**:** Effect of the presence of strongly or weakly relevant pretuning capability on fine-tuning performance on using η_M and η_S. Test sets with and without the spurious correlations are used for evaluation. **Observation:** The convergence time for learning the fine-tuning task in the absence of strongly relevant capability is higher as compared to when the strongly relevant is present in the model. The time further increases if spurious correlations are present in the fine-tuning set. However, in the presence of spurious correlations, the convergence time to learn the spurious correlation is small and is possible even on using the learning rate η_S. Using η_S is unable to yield learning of the fine-tuning task if a weakly relevant capability is present in the model.

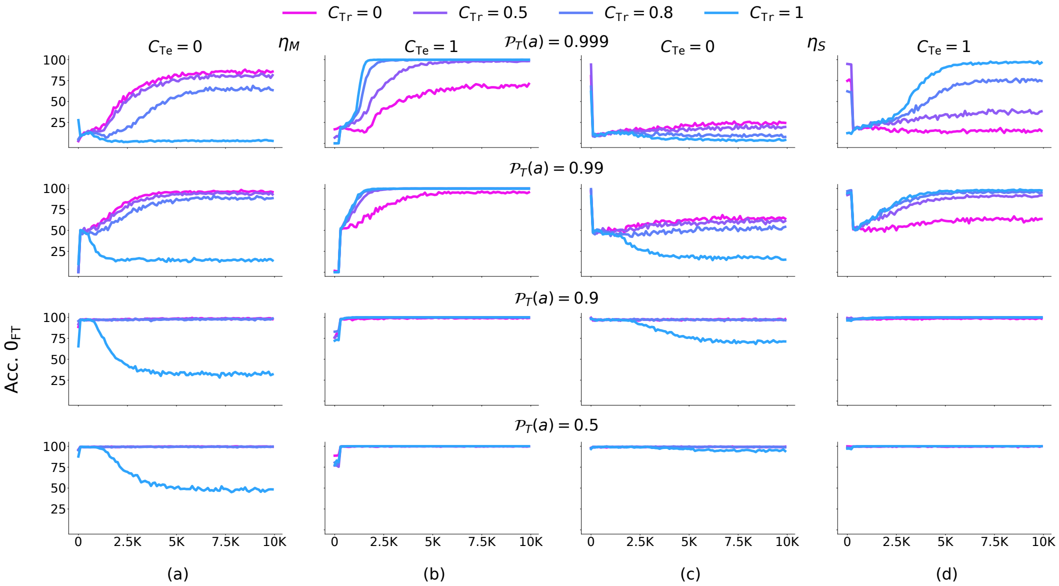


Figure 62: **Index of Occurrence Task,** $n_{iters} = 200K$**:** The settings are consistent with Fig. 61.

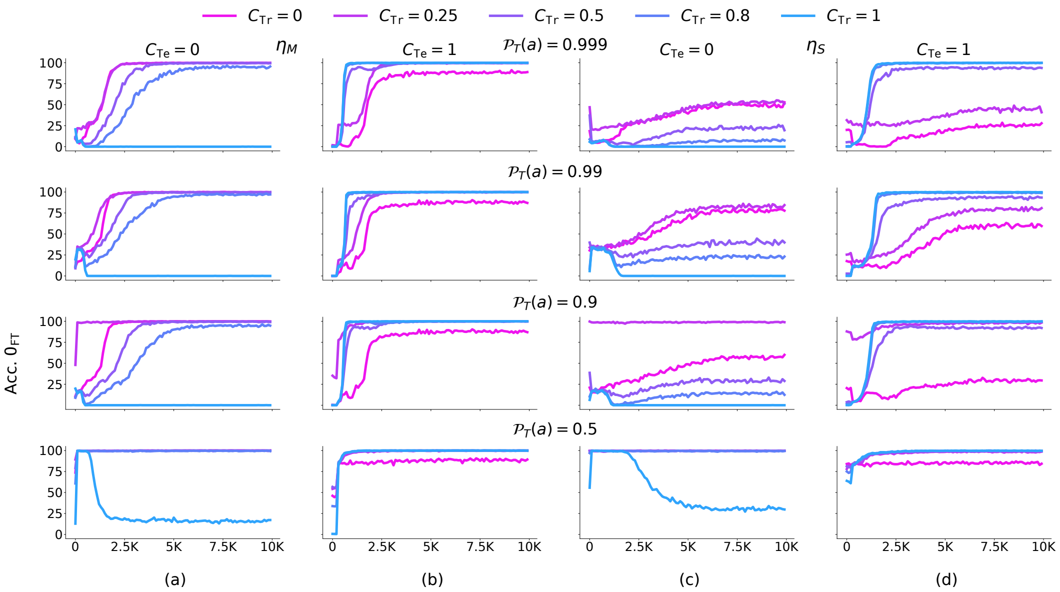


Figure 63: **Counter Task,** $n_{iters} = 50K$**:** Effect of the presence of strongly or weakly relevant pretuning capability on fine-tuning performance on using η_M and η_S. Test sets with and without the spurious correlations are used for evaluation. The observations are consistent with Fig. 61.

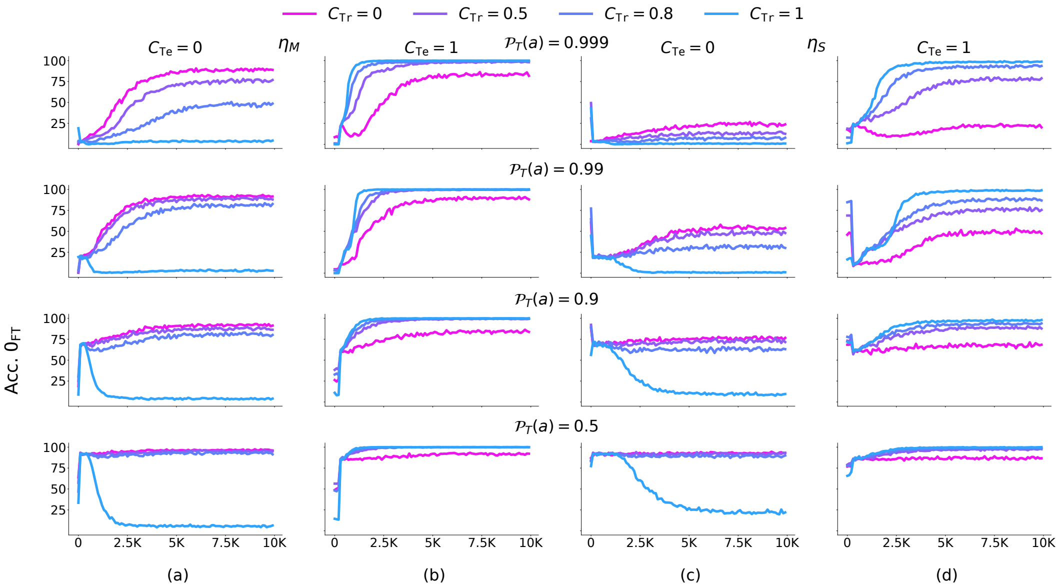


Figure 64: **Index of Occurrence Task,** $n_{iters} = 50K$**:** The settings are consistent with Fig. 63.

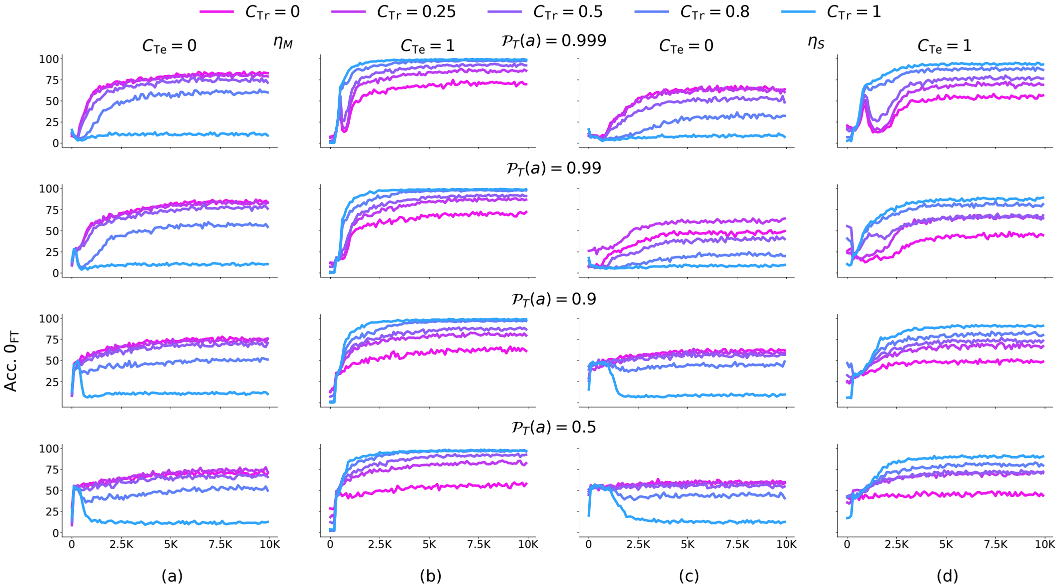


Figure 65: **Counter Task,** $n_{iters} = 10K$**:** Effect of the presence of strongly or weakly relevant pretuning capability on fine-tuning performance on using η_M and η_S. Test sets with and without the spurious correlations are used for evaluation. The observations are consistent with Fig. 61.

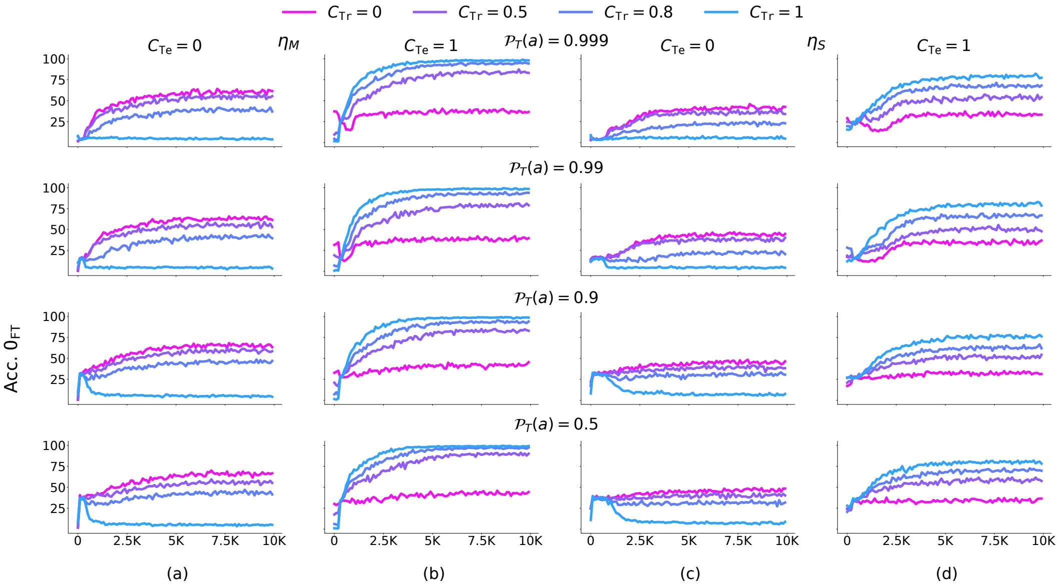


Figure 66: **Index of Occurrence Task,** $n_{iters} = 10K$**:** The settings are consistent with Fig. 65.

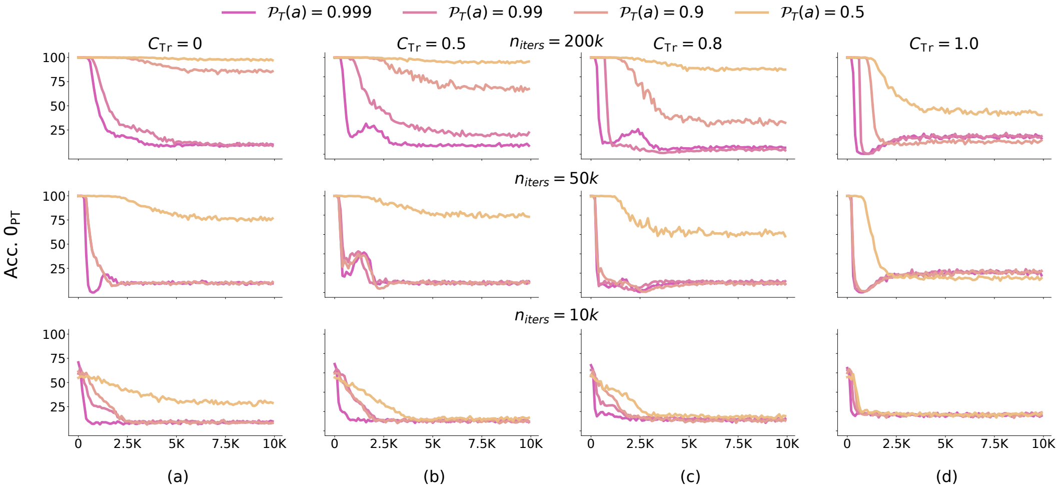


Figure 67: **Counter Task,** η_M**,** $C_{\mathtt{Te}} = 0$ **:** Learning of the wrapper in presence of different fraction of spuriously correlated data, values of $\mathcal{P}_{\mathtt{T}}$ $(\mathtt{a})$ during pre-training, and training iterations. **Observation:** Using a higher fraction of spuriously correlated data in the fine-tuning set (higher value of $C_{\mathtt{Tr}}$) leads to faster degradation in the pre-training accuracy. Further this degradation is even faster in presence of weakly relevant capability.

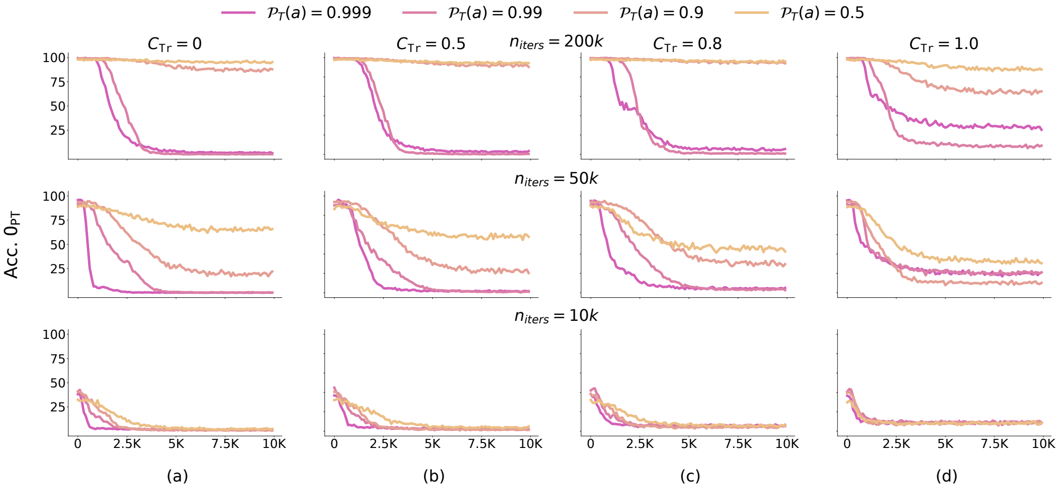


Figure 68: **Index of Occurrence Task, Medium LR,** $C_{\mathtt{Te}} = 0$**:** The settings are consistent with Fig. 67.

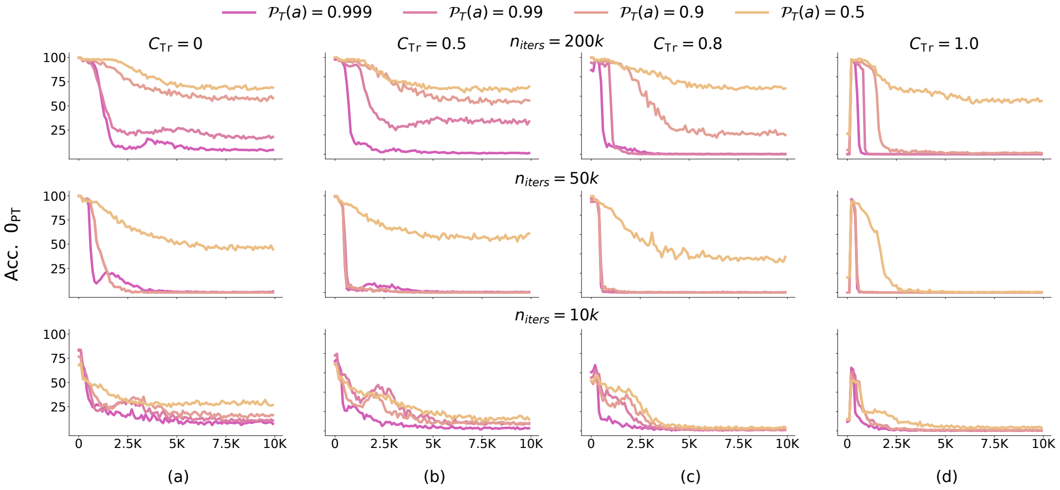


Figure 69: **Counter Task,** η_M**,** $C_{\mathtt{Te}} = 1$ **:** Learning of the wrapper in presence of different fraction of spuriously correlated data, values of $\mathcal{P}_{\mathtt{T}}(\mathtt{a})$ during pre-training, and training iterations. **Observation:** Using a higher fraction of spuriously correlated data in the fine-tuning set (higher value of $C_{\mathtt{Tr}}$) leads to faster degradation in the pre-training accuracy. Further this degradation is even faster in presence of weakly relevant capability.

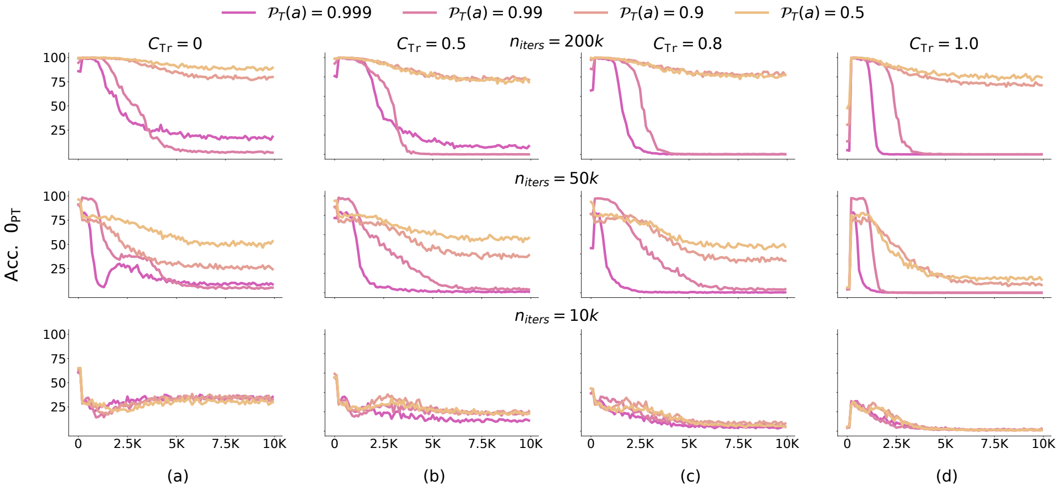


Figure 70: **Index of Occurrence Task, Medium LR,** $C_{\mathtt{Te}} = 1$ **:** The settings are consistent with Fig. 69.

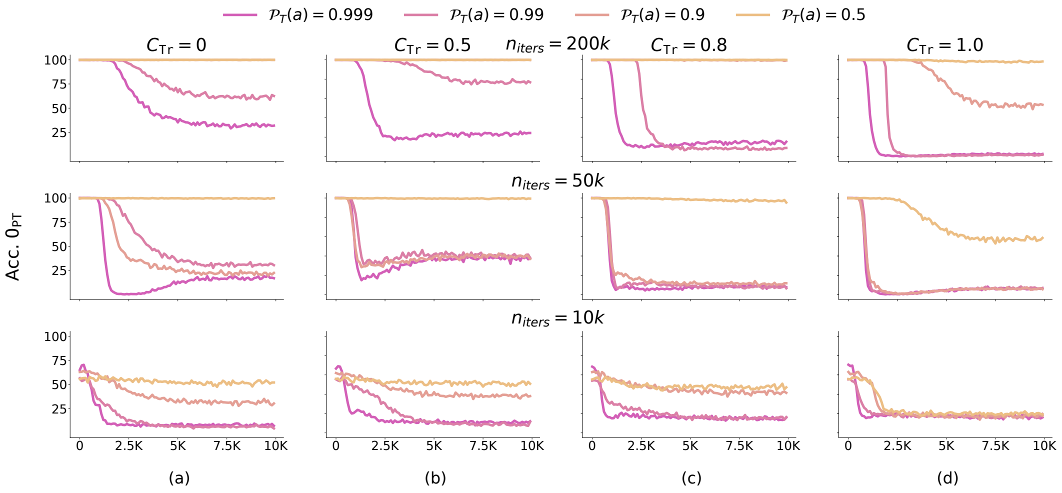


Figure 71: **Counter Task,** η_S, $C_{\mathtt{Te}} = 0$ **:** Learning of the wrapper in presence of different fraction of spuriously correlated data, values of $\mathcal{P}_{\mathtt{T}}$ (a) during pre-training, and training iterations. **Observation:** The observations are consistent with Fig. 67

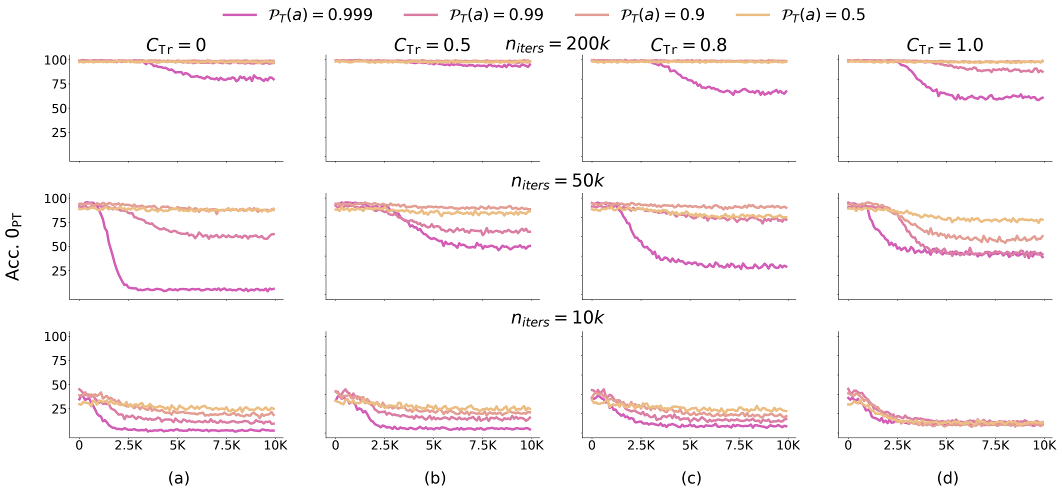


Figure 72: **Index of Occurrence Task,** η_S, $C_{\mathtt{Te}} = 0$ **:** The settings are consistent with Fig. 71

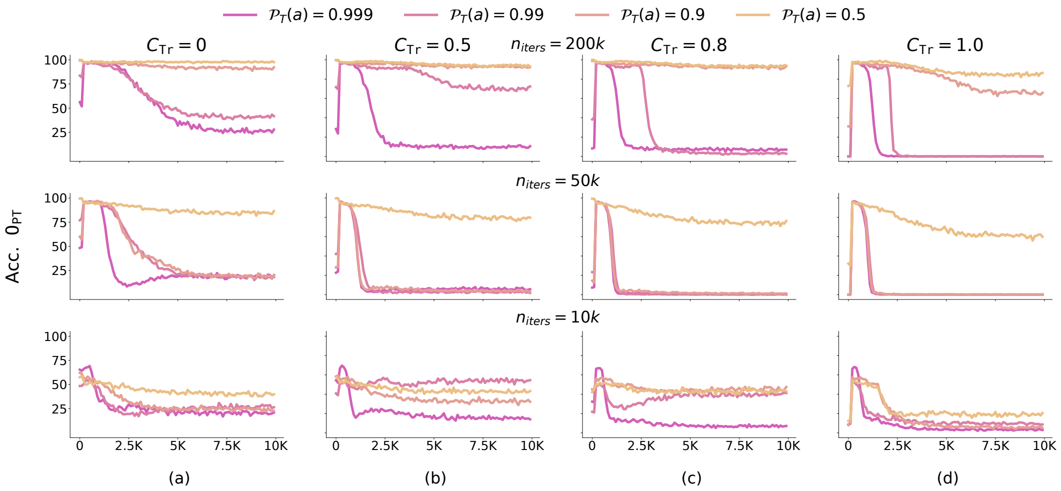


Figure 73: **Counter Task,** η_S**,** $C_{\mathtt{Te}} = 1$ **:** Learning of the wrapper in presence of different fraction of spuriously correlated data, values of $\mathcal{P}_{\mathrm{T}}$ (a) during pre-training, and training iterations. **Observation:** The settings are consistent with Fig. 69

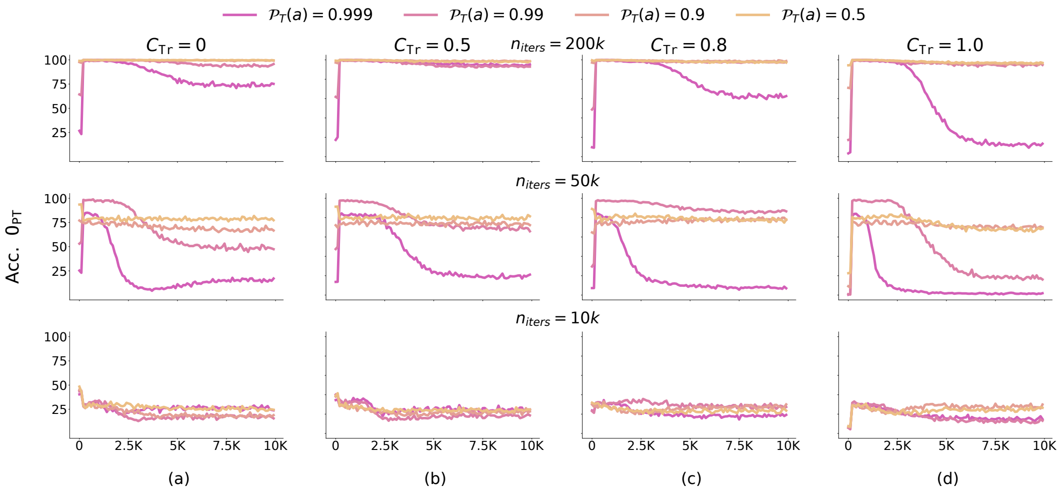


Figure 74: **Index of Occurrence Task,** η_S**,** $C_{\mathtt{Te}} = 1$ **:** The settings are consistent with Fig. 73

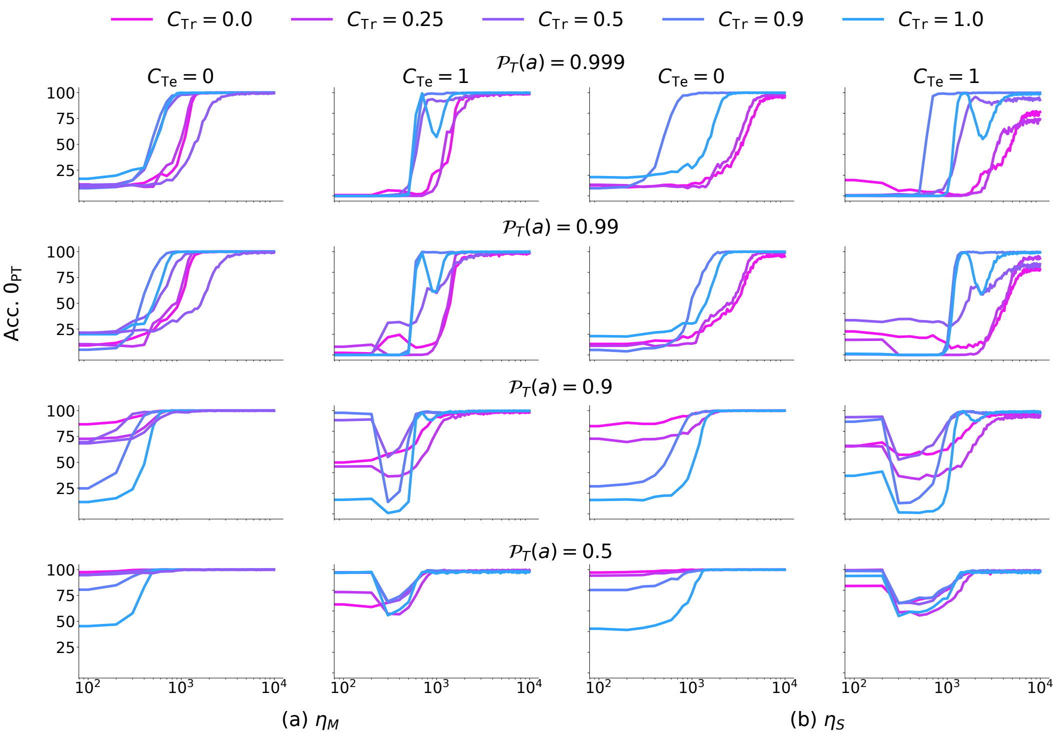


Figure 75: **Counter Task,** $n_{iters} = 200K$**: Reverse Fine-tuning** on weakly and strongly relevant capability fine-tuned models. Medium and small learning rates are used for reverse fine-tuning in the presence of different degrees of spuriously correlated data-points present in the train-set. The fine-tuned model was fine-tuned using Large LR. **Observation:** When the model possesses weakly relevant capability, the convergence time is lower for models fine-tuned on dataset with spurious correlations. If the model possesses strongly relevant capability, this difference is less. The "revival" of pre-training capability is observed for all values of $C_{\mathtt{Tr}}$. Even though fine-tuning was done using a large learning rate of 10^{-4}, capability revival is possible even on using a small learning rate.

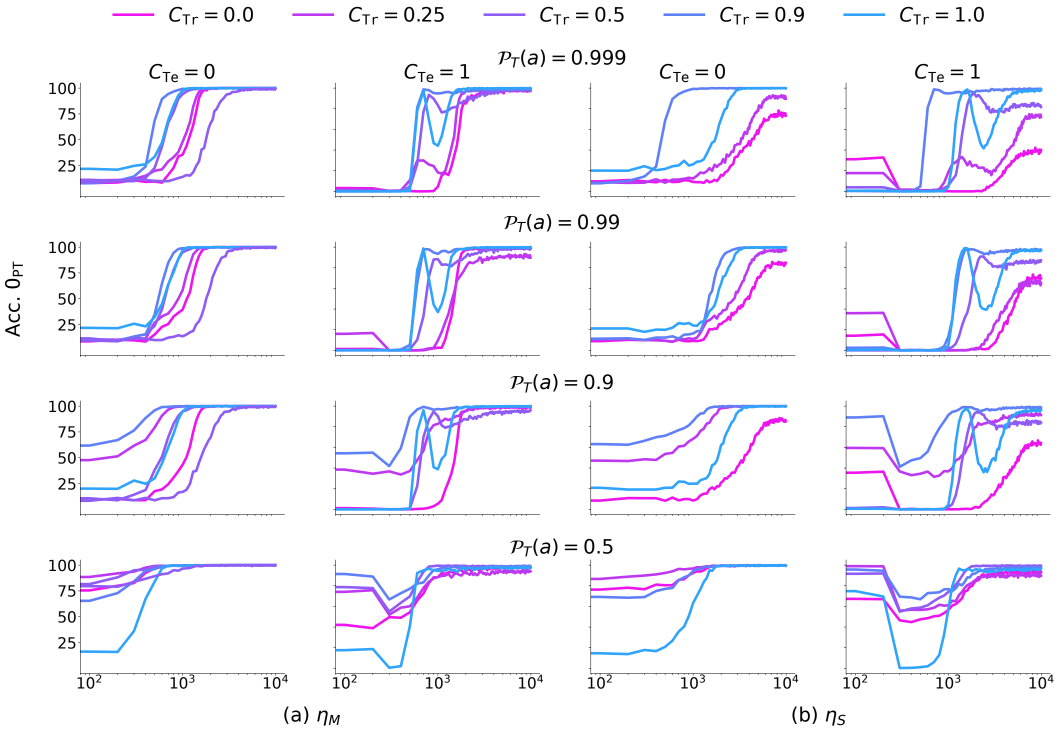


Figure 76: **Counter Task, $n_{iters} = 50K$: Reverse Fine-tuning** on weakly and strongly relevant capability fine-tuned models. η_M and η_S are used for capability reverse fine-tuning in the presence of different fraction of spuriously correlated data-points present in the train-set. The fine-tuned model was fine-tuned using Large LR. **Observation:** The observations are consistent with Fig. 75

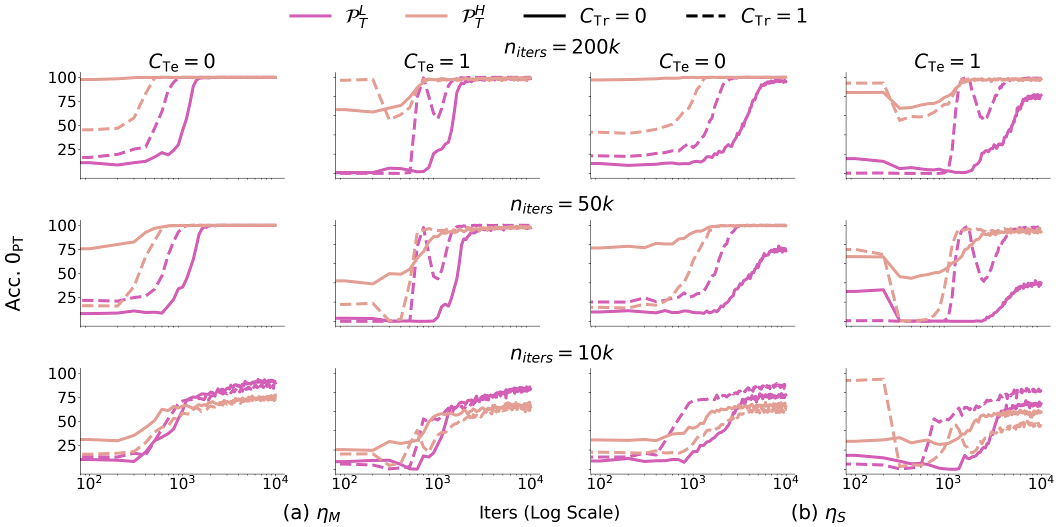


Figure 77: **Counter Task: Reverse fine-tuning analysis for different pre-training iterations. Observation:** Capability Revival is seen for models pre-trained with different number of iterations.

J.1 Pruning Analysis

In this section, we present detailed results on pruning analysis of the PCFG setup on both counting and index of occurrence tasks. We provide an exhaustive evaluation in Fig. 78, 81, 83, 85 and 87 for the Counter task and Fig. 79, 82, 84, 86 and 88 for the index of occurrence task.

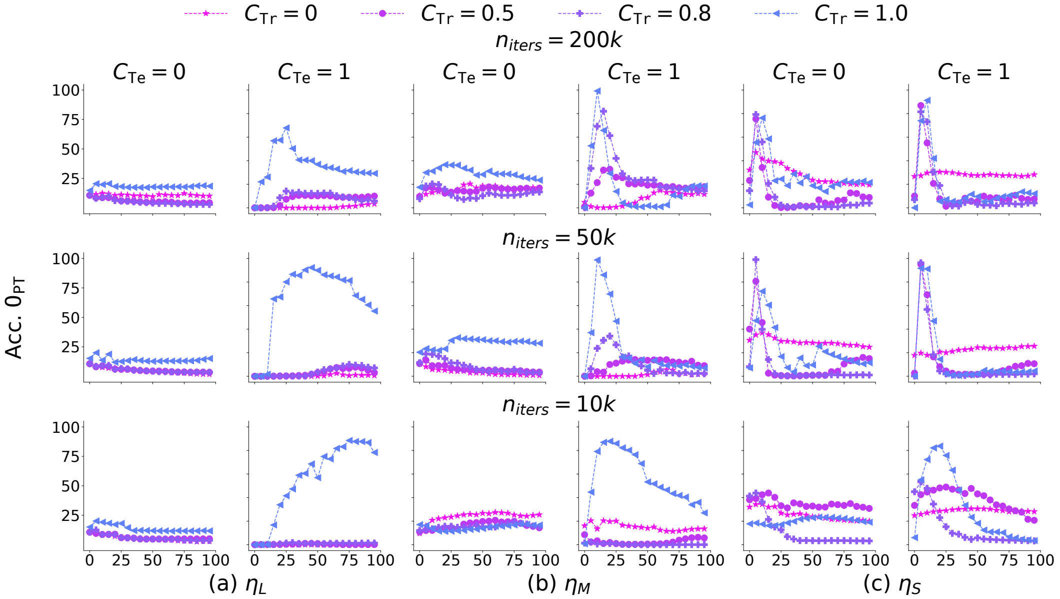
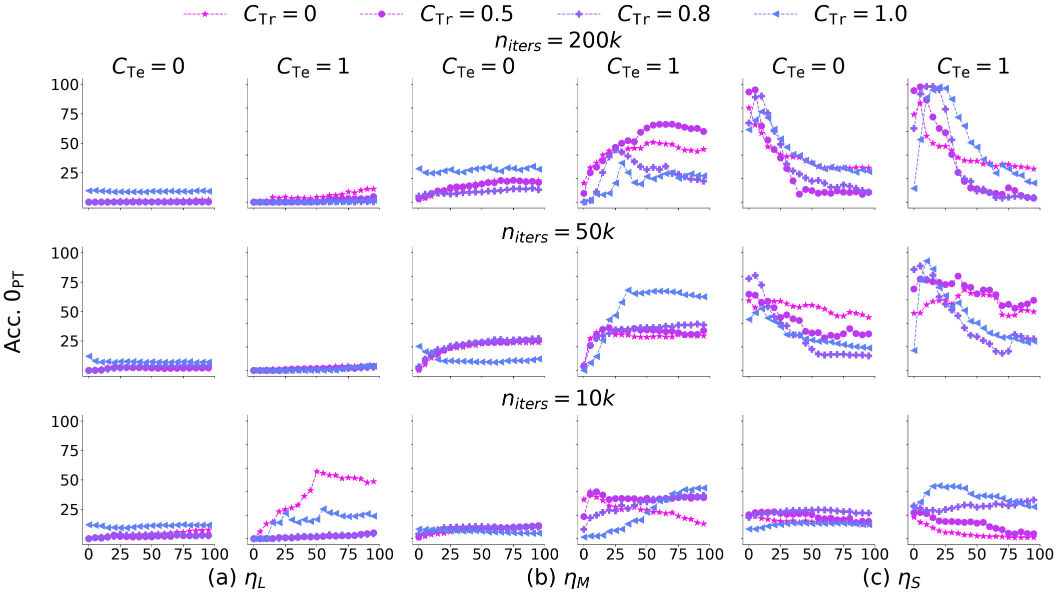
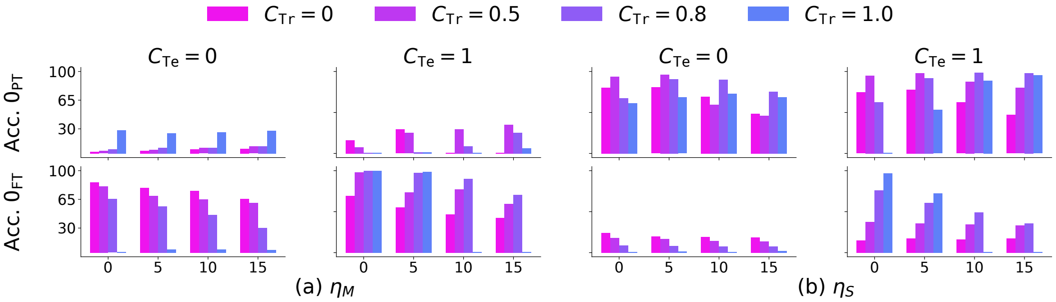


Figure 78: **Counter task, $\mathcal{P}_{\text{T}}(\text{a}) = 0.999$, Pruning Analysis:** Revival of pre-training capability analysis for different learning rates, pre-training iterations and different values of C_{Tr}. **Observation:** (a) On using η_L the model learns the wrapper only when fine-tuning on $C_{\text{Te}} = 1$. This wrapper is learned only on fine-tuning set with the spurious correlations . (b) On using η_M the model learns the wrapper on fine-tuning with smaller values of C_{Te} as well. However, still this wrapper is learned only on fine-tuning set with the spurious correlations. (c) On using η_S the model learns the wrapper for all values of C_{Te} and for all the data samples.



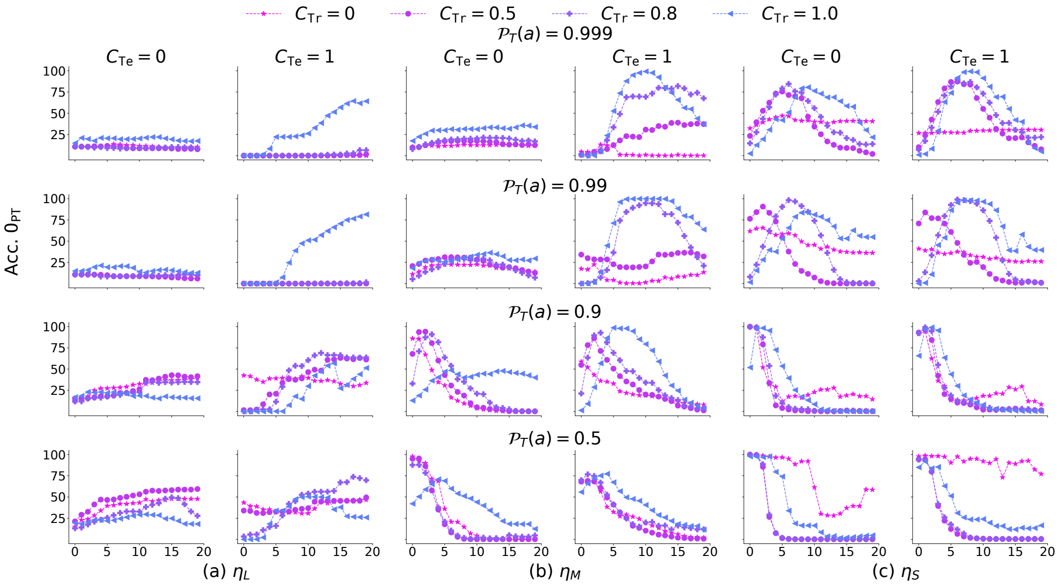
(a) η_L (b) η_M (c) η_S

Figure 79: **Index of Occurrence task,** $\mathcal{P}_{\mathrm{T}}(\mathrm{a}) = 0.999$**, Pruning Analysis:** The settings are consistent with Fig. 78



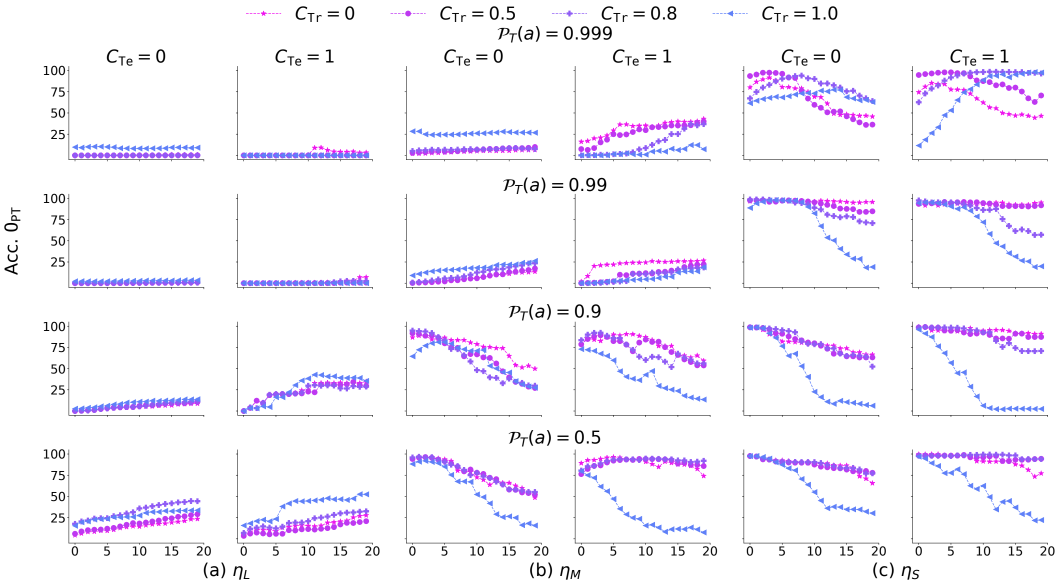
(a) η_M (b) η_S

Figure 80: **Index of Occurrence task,** $\mathcal{P}_{\mathrm{T}}(\mathrm{a}) = 0.999$**, Pruning Analysis:** The settings are consistent with Fig. 3



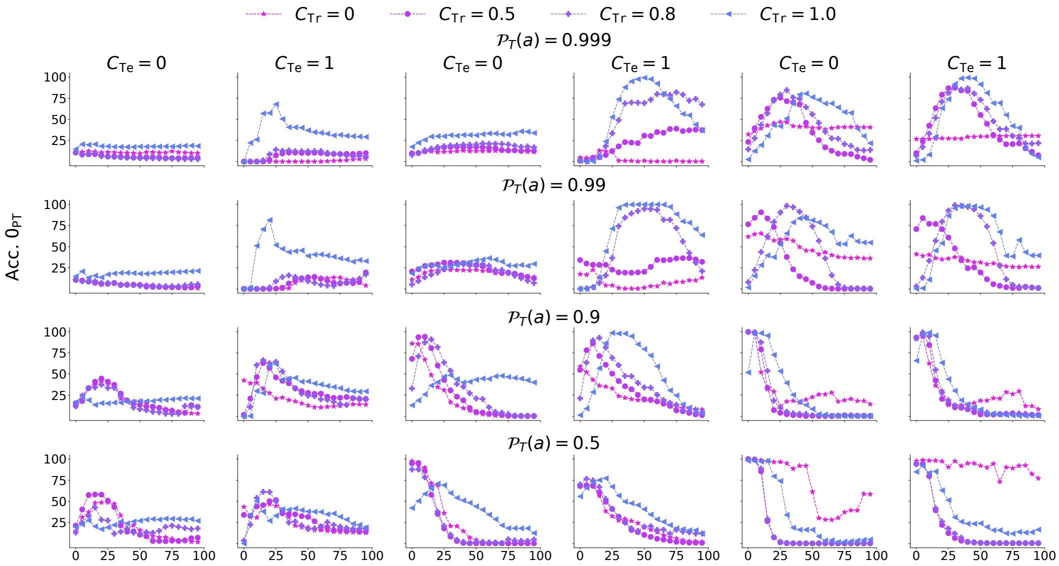
(a) η_L (b) η_M (c) η_S

Figure 81: **Counter task, $n_{iters} = 200K$, Pruning Analysis:** Revival of pre-training capability analysis for different learning rates, weakly and strongly relevant capability fine-tuned models, and different values of C_{Tr}. **Observation:** Learning of the wrapper is possible for weakly as well as strongly relevant capability pre-trained models.



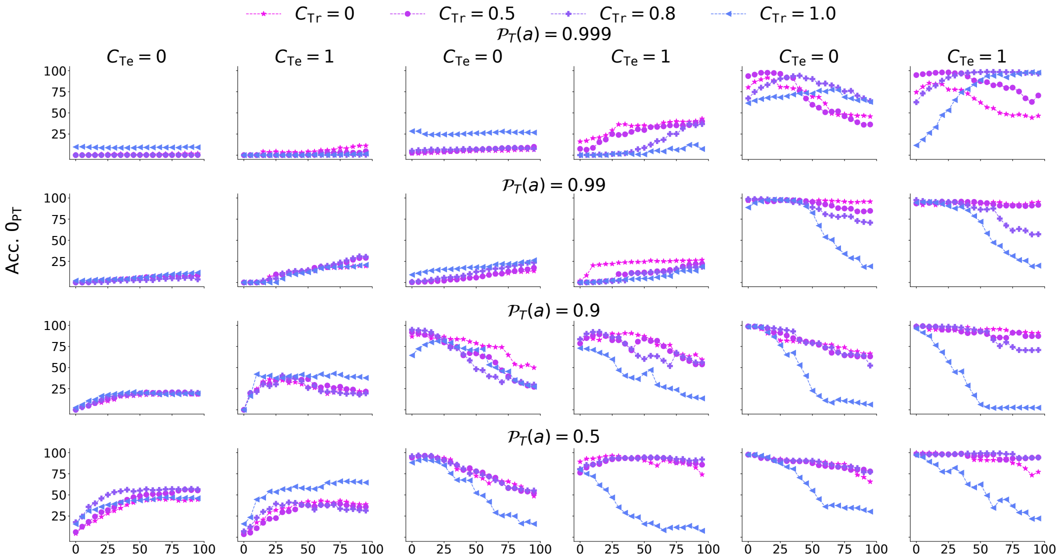
(a) η_L (b) η_M (c) η_S

Figure 82: **Index of Occurrence task, $n_{iters} = 200K$, Pruning Analysis:** The settings are consistent with Fig. 81



(a) η_L (b) η_M (c) η_S

Figure 83: **Counter task,** $n_{iters} = 200K$ **Pruning Analysis:** Revival of pre-training capability analysis for different learning rates, weakly and strongly relevant capability fine-tuned models and different values of $C_{\texttt{Tr}}$. Here larger number of neurons are pruned as compared to Fig. 81.



(a) η_L (b) η_M (c) η_S

Figure 84: **Index of Occurrence task,** $n_{iters} = 200K$**, Pruning Analysis:** Here larger number of neurons are pruned. The settings are consistent with Fig. 83

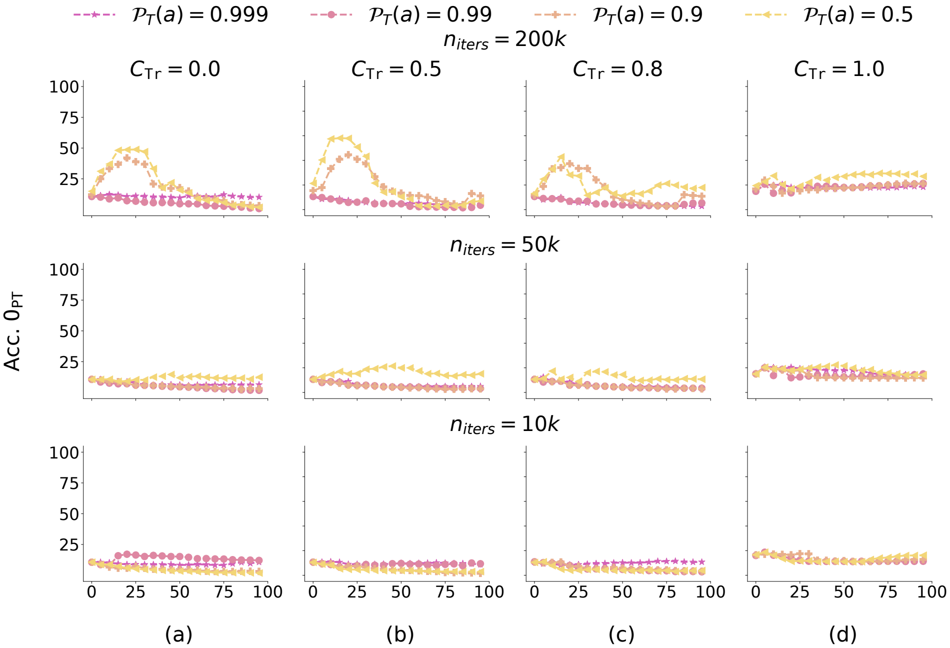


Figure 85: **Counter task,** η_L**,** $C_{\texttt{Te}} = 0$**, Pruning analysis:** Effect of strongly and weakly relevant capabilities for different number of pre-training iterations and different values of $C_{\texttt{Tr}}$ **Observation:** Fine-tuning a model with strongly relevant capability leads to learning of an "inhibitor" on its pre-training capability, i.e., a wrapper that disallows use of the pretraining capability. Revival of the pre-training capability is partly possible on pruning, if the model has strongly relevant capability and it was fine-tuned on dataset without spurious correlations.The inhibitor is mainly learned for the $200K$ iteration pre-trained model.

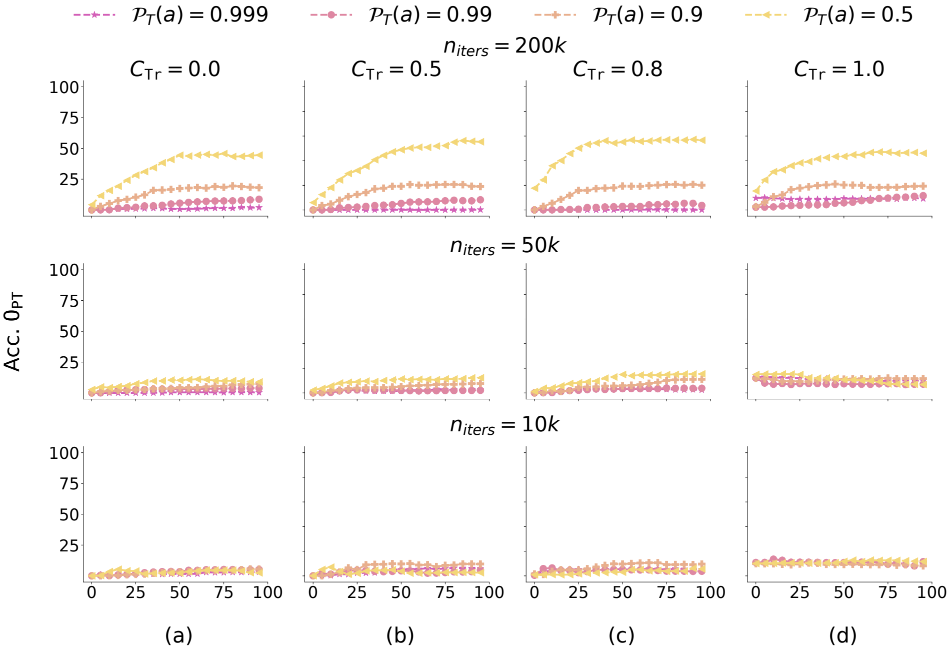


Figure 86: **Index of Occurrence task,** η_L**,** $C_{\texttt{Te}} = 0$**, Pruning analysis:** The settings are consistent with Fig. 85

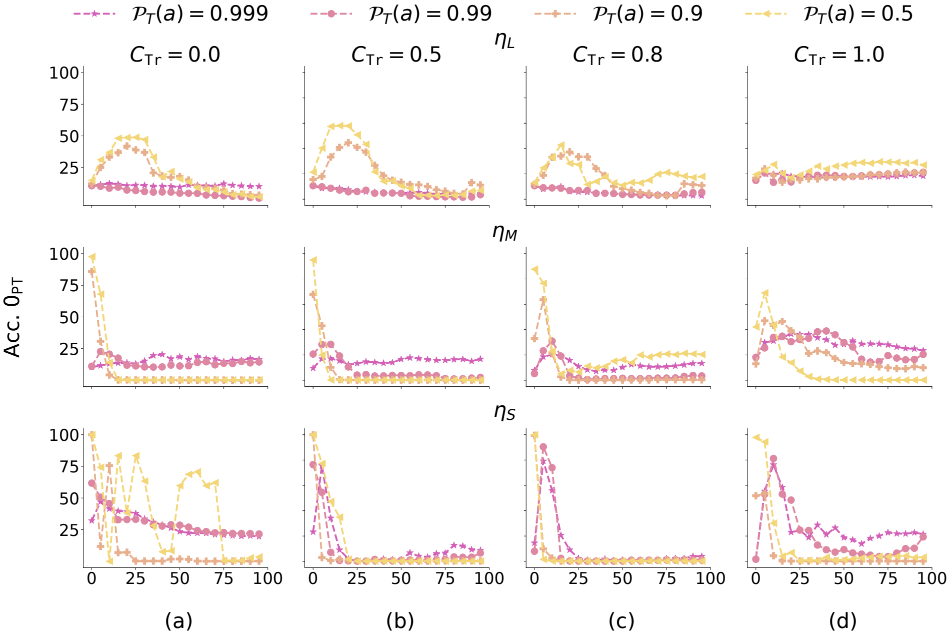


Figure 87: **Counter task,** $n_{iters} = 200K$**,** $C_{\text{Te}} = 0$**, Pruning analysis:** Effect of the strongly and weakly relevant capabilities for different number of pre-training iterations and different values of fraction of spurious correlations present in the fine-tuning dataset. **Observation:** Learning of the inhibitor is observed on using η_L.

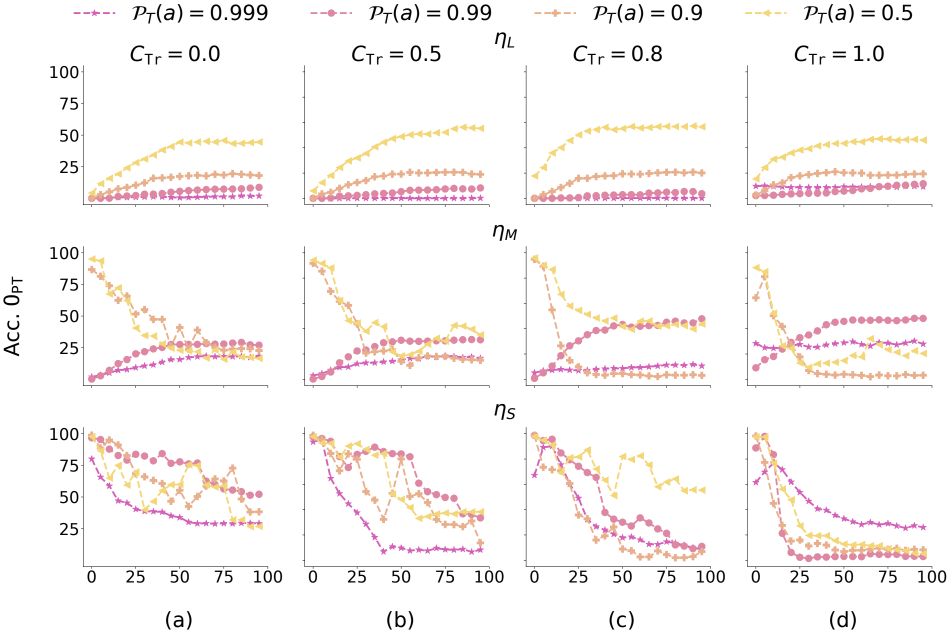


Figure 88: **Index of Occurrence task,** $n_{iters} = 200K$**,** $C_{\text{Te}} = 0$ **Pruning analysis:** the settings are consistent with Fig. 87

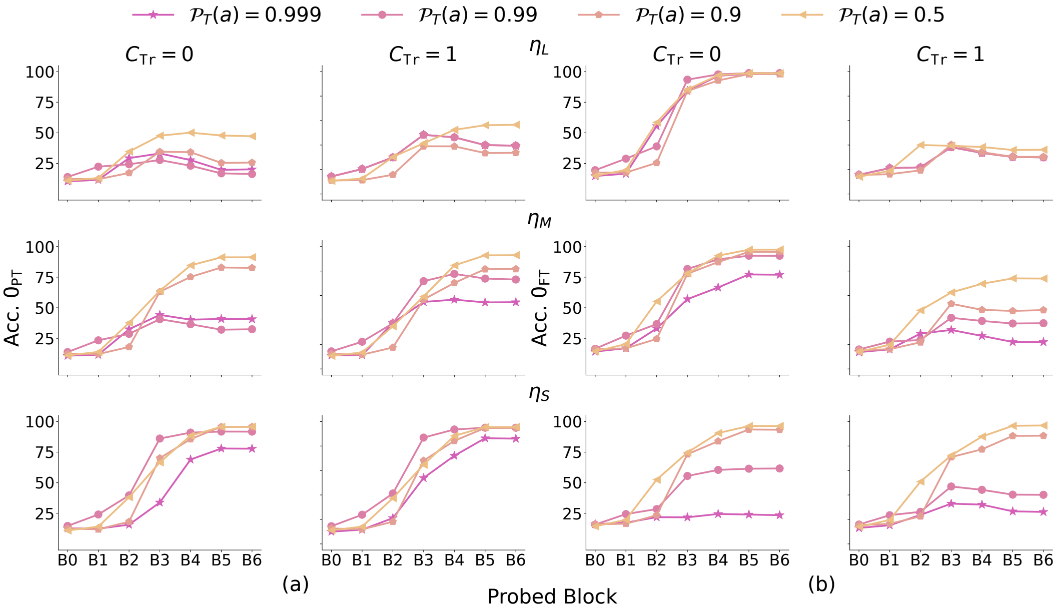


Figure 89: **Counter task,** $n_{iters} = 200K$ **,** $C_{\texttt{Te}} = 0$**, Probing analysis:** The effect of different values of learning rate, weakly and strongly relevant capabilities is shown. **Observation:** Using η_L hampers the pre-training capability to count a especially when the probed model has a weakly relevant capability. The performance on the pre-training task of counting a's continues to be high, especially with η_S. The accuracy of counting b's shows that fine-tuning capability is learned on using η_L. On using η_S, models with weakly relevant capabilities are not able to learn the fine-tuning capability well.

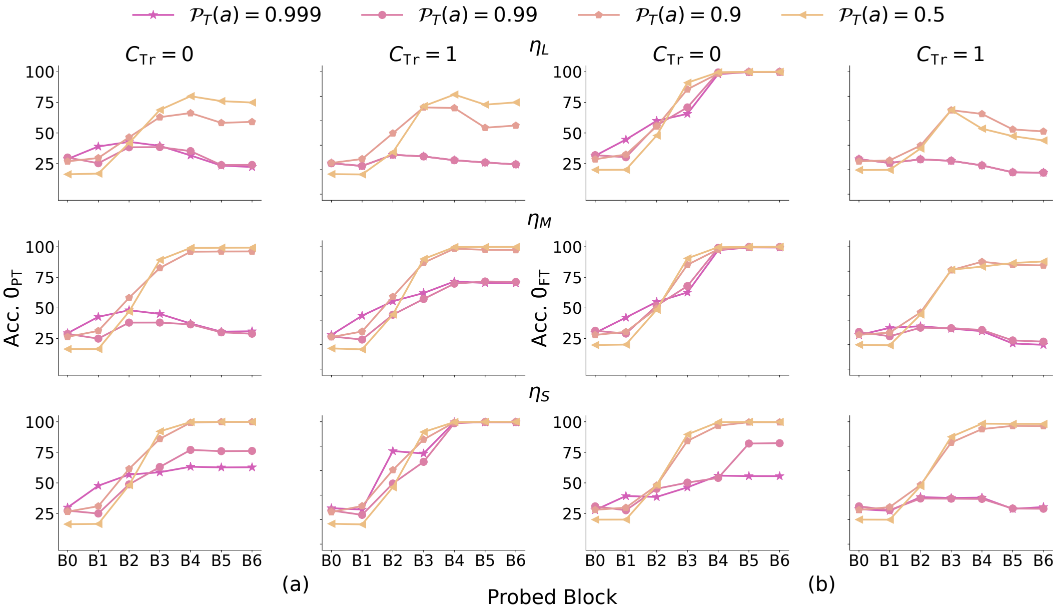


Figure 90: **Index of Occurrence task,** $n_{iters} = 200K$ **,** $C_{\texttt{Te}} = 0$**, Probing analysis:** The settings are consistent with Fig. 89

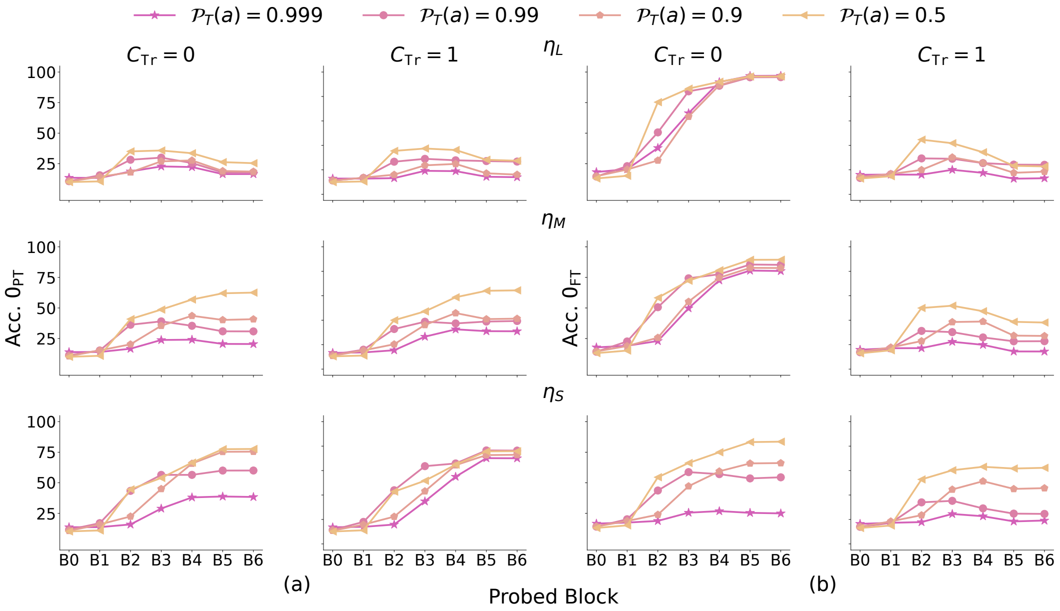


Figure 91: **Counter task,** $n_{iters} = 50K$**,** $C_{\mathrm{Te}} = 0$**, Probing analysis:** The observations are consistent with Fig. 89

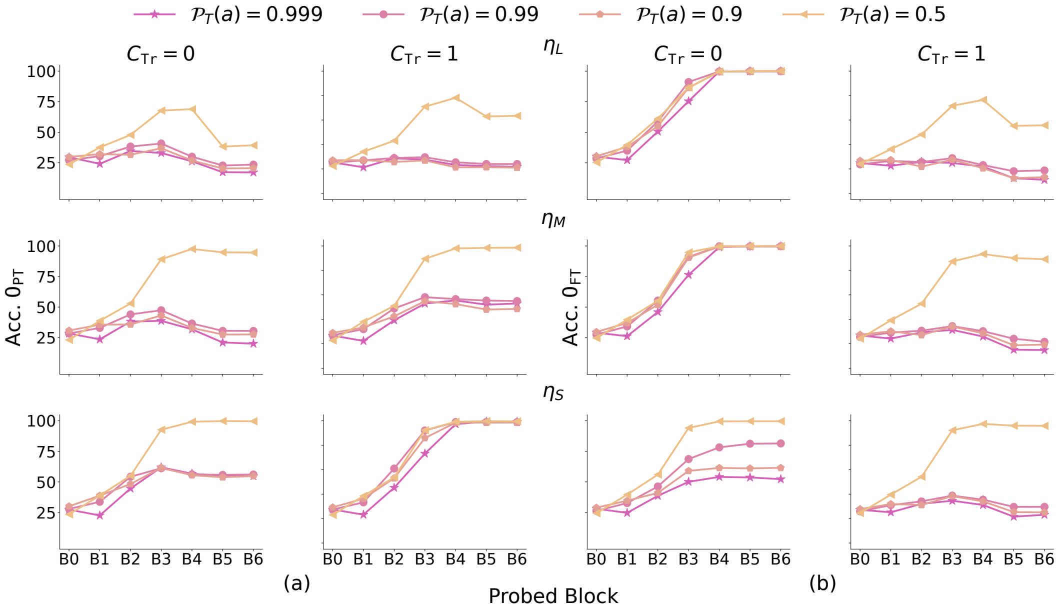


Figure 92: **Index of Occurrence task,** $n_{iters} = 50K$**,** $C_{\mathrm{Te}} = 0$**, Probing analysis:** The settings are consistent with Fig. 91

J.2 Probing Analysis

In this section, we present detailed results on probing analysis of the PCFG setup on both counting and index of occurrence tasks. We provide an exhaustive evaluation in Fig. 91, 94, 96 for the Counter task and Fig. 92, 95, 97 for the index of occurrence task.

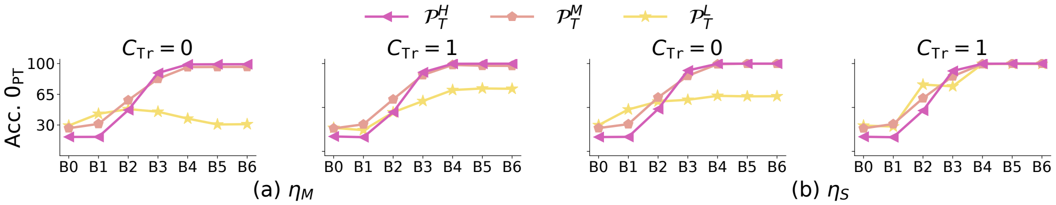


Figure 93: **Index of Occurrence task,** $n_{iters} = 200K$ **,** $C_{\texttt{Te}} = 0$**, Probing analysis:** The settings are consistent with Fig. 8

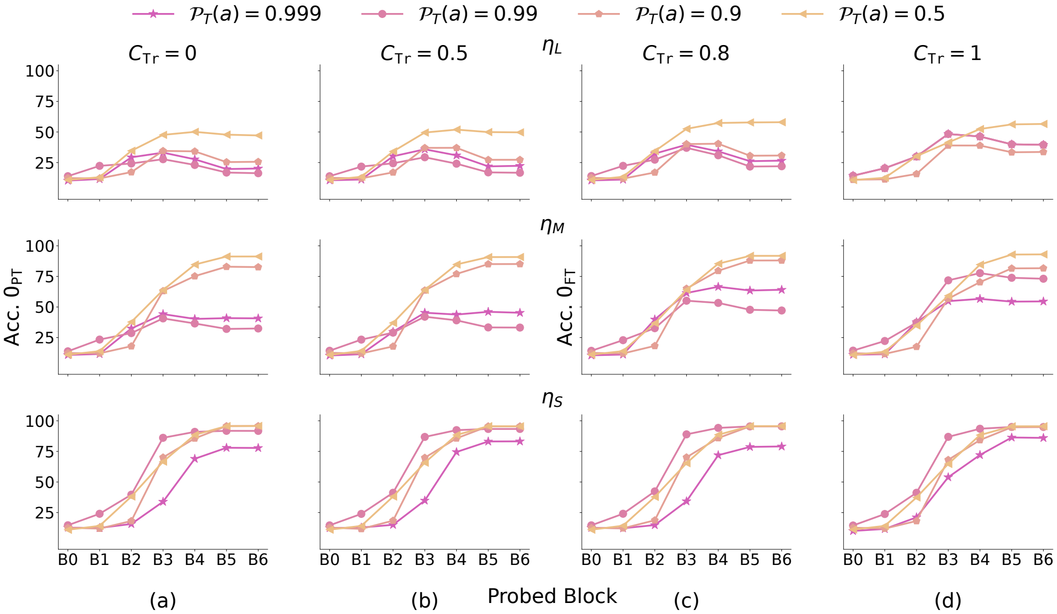


Figure 94: **Counter task,** $n_{iters} = 200K$ **,** $C_{\texttt{Te}} = 0$**, Probing analysis. Observation:** With an increase in $C_{\texttt{Tr}}$, the accuracy on counting a's also increases for both weakly as well as strongly relevant capability models.

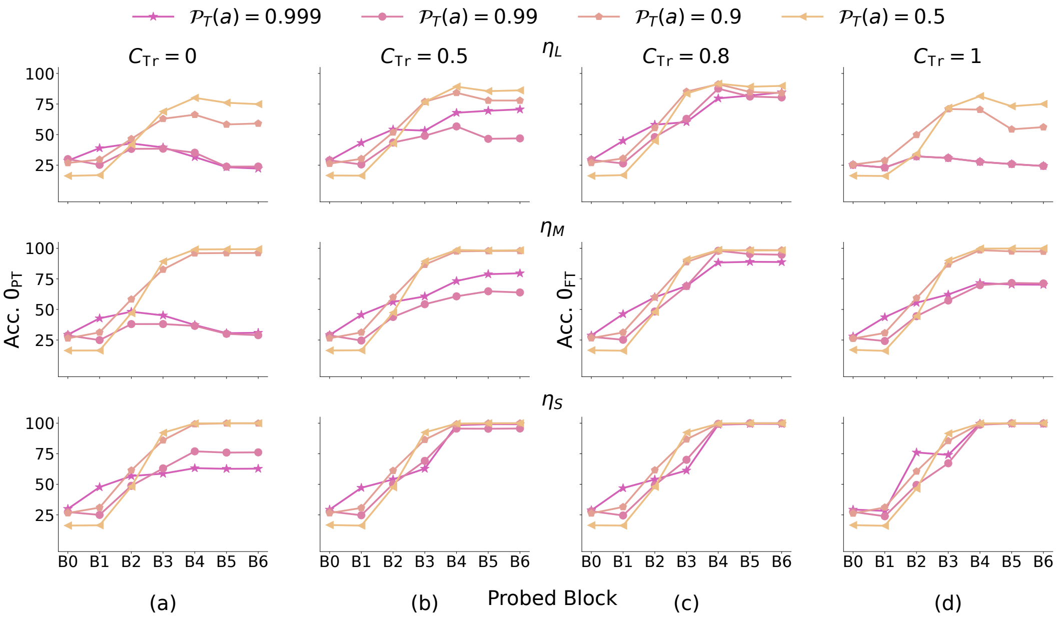


Figure 95: **Index of Occurrence task,** $n_{iters} = 200K$ **,** $C_{\mathtt{Te}} = 0$**, Probing analysis:** The settings are consistent with Fig. 94

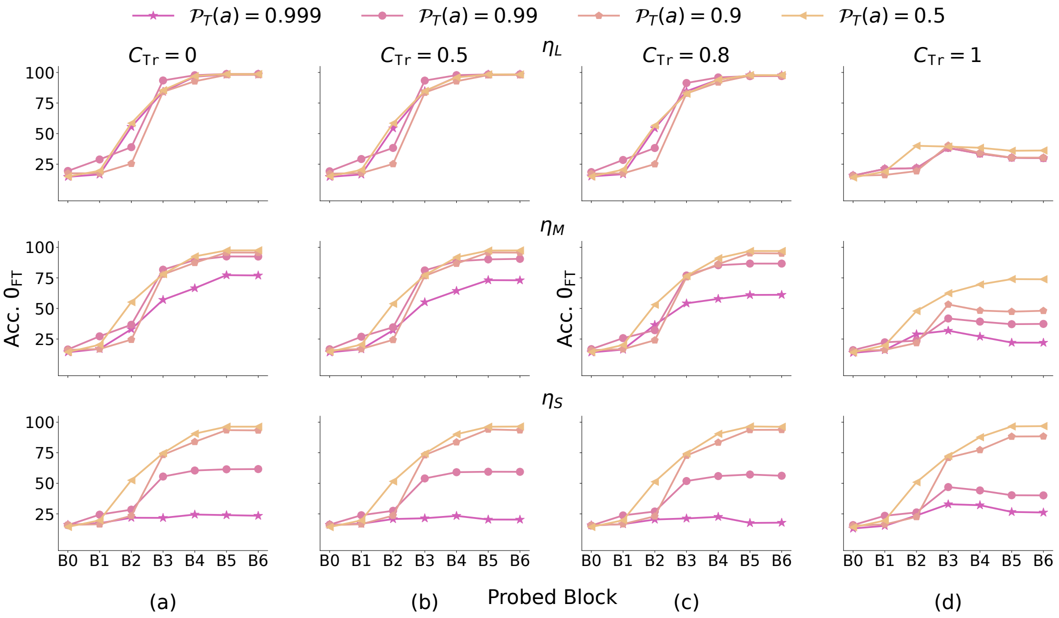


Figure 96: **Counter task,** $n_{iters} = 50K$ **,** $C_{\mathtt{Te}} = 0$**, Probing analysis. Observation:** With an increase in $C_{\mathtt{Tr}}$, the accuracy on counting b's also decreases for both weakly as well as strongly relevant capability models.

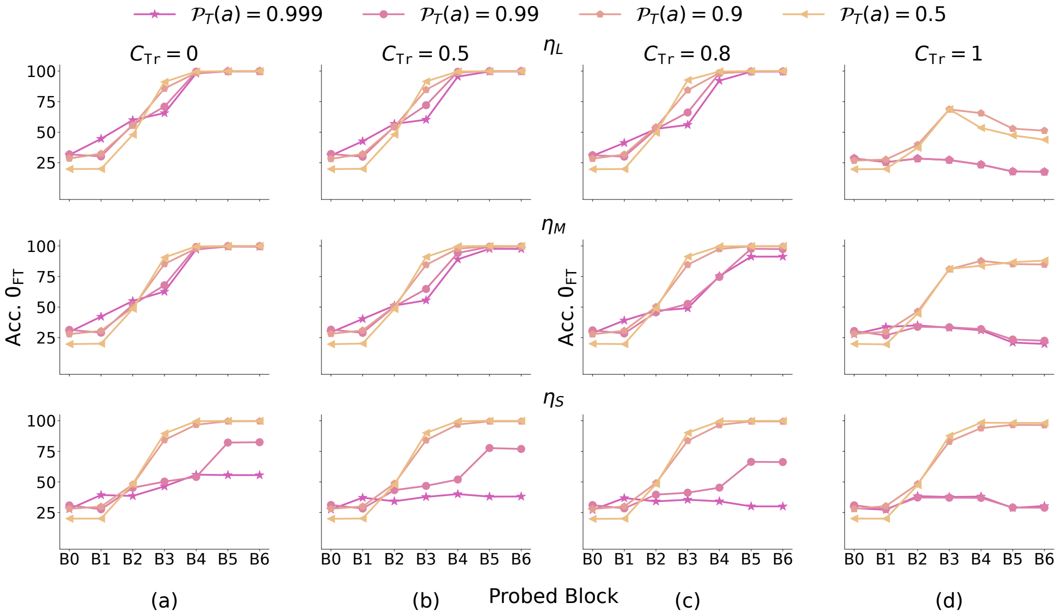


Figure 97: **Index of Occurrence task,** $n_{iters} = 50K$ **,** $C_{\texttt{Te}} = 0$**, Probing analysis:** The settings are consistent with Fig. 96