UNIVERSITY OF CAMBRIDGE
Department of Engineering

Can an imitation learning agent inherit the goals of an expert:

A case study of the traveling salesperson problem

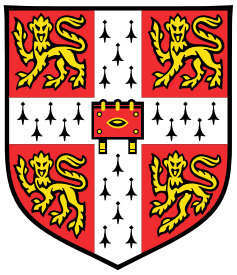
Author Name: Ognjen Stefanović
Supervisor: Prof. David Krueger
Date: 27.05.2024.

I hereby declare that, except where specifically indicated, the work submitted herin is my own original work.

Signed 

date 27.05.2024.

Can an imitation learning agent inherit the goals of an expert: A case study of the traveling salesperson problem



Ognjen Stefanović

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Engineering

Trinity College May 2024

Acknowledgements

I would like to express my deepest gratitude to Alan Chan and Ekdeep Singh Lubana, who supervised me throughout my research project. They were always available and quick to respond, providing invaluable insight and suggestions for the next steps. Their patience during times when the project progressed slowly was outstanding, and for that, I am deeply grateful.

I would also like to extend my deepest gratitude to Prof. David Krueger, who enabled me to undertake the project in his group and provided consistently useful feedback.

Special thanks are due to the University of Cambridge Research Computing Services for granting me access, and to the Cambridge HPC staff for ensuring the smooth operation of the servers and program jobs.

Abstract

Language models (LMs) have been argued to possess beliefs, desires, and intentions which influence their outputs, resulting in arguments that these models can learn and pursue goals. This dissertation questions this narrative, asking whether LMs can learn to model goals and behave in a goal-directed manner when trained via imitation learning under an expert agent. Specifically, this case study focuses on the traveling salesperson problem (TSP), where the goal is unequivocally defined: to find the shortest route to visit each city and return to the start. The TSP setup used has five cities named `A`, `B`, `C`, `D` and `E`, which are placed on a 2D integer grid of shape 6×6. All possible city placements and their respective optimal paths are determined and used to create various training and evaluation sets. Each TSP example, along with its optimal traversal, is presented to the model as a sequence of characters and letters.

Under this clear and well-defined setup, the notion of goals and what it means for a LM to act in a goal-directed manner is explored. The definition of goal-directedness assumed here is that the model learns to perform a task and can exhibit *adaptivity*. Model adaptivity means the model will take "active" actions to complete the task even when there is a systematic difference between the train and test set. The experimental setups ensure that the goals in the evaluation sets are out-of-distribution in distinct ways while maintaining the correctness of the goals in the training set. This approach allows us to investigate generalisation in different problem setups.

The first experiment shows the model's ability to generalise to a previously unseen coordinate, achieving performance on par with the baseline. Additionally, it reveals that introducing this coordinate into the training set a few times does not affect performance.

Following this, tougher generalisation problems with more unseen coordinates at the board centre are posed. The model is evaluated on various problems containing cities at these previously unseen coordinates. The primary findings from these experiments indicate that the model acts in a goal-directed manner, successfully solving TSP with cities situated on previously unseen grid segments.

The goal-directedness of the model is also examined in presence of two different types of noise in the training sets. Random noise is introduced to the optimal salesperson traversing

by randomly swapping two out of the last four cities in the optimal traversing. Systematic noise is introduced by swapping cities `B` and `C` in the optimal salesperson traversing. Both noise types make the traversings suboptimal.

When random noise is introduced, the model demonstrates the ability to pursue the goal despite substantial noise. With systematic noise, the model tends to follow the goal most prevalent in the training dataset. *By doing so, the model is able not only to match but also to exceed the training agent's performance.* Additionally, we notice that the model generalises overconfidently on the evaluation set, finding optimal paths even when trained with significant amounts of systematic or random noise.

Finally, a mechanistic interpretability analysis of model activations in a simplified setting (for tractability purposes) with various inputs is conducted to try to uncover any emergent goal representations within the model weights. While the findings from this analysis are encouraging, there is still potential for future work.

Table of contents

Chapter 1

Introduction

Language models (LMs) have been argued to possess beliefs, desires, and intentions which influence their outputs, resulting in arguments that these models can learn and pursue goals. This dissertation questions this narrative, asking whether LMs can learn to model goals and behave in a goal-directed manner when trained via imitation learning under an expert agent.

We start by defining what is an agent. In Intentional Systems Theory, Dennett (2009) defines a rational agent to be an entity "who governs its ‘choice’ of ‘action’ by a ‘consideration’ of its ‘beliefs’ and ‘desires’". In other words, an agent is an entity whose behaviour can be explained by attributing it beliefs, desires and the ability to reason. Shimi et al. (2021) say that desires correspond to goals and beliefs correspond to models of the world. World models are understandable models of the process which produce training sequences (Li et al. (2023)).

Orseau et al. (2018) go a step further and provide an definition for an agent via ideas grounded in computational theory, which categorises a system to be an agent in terms of the function it optimises. So one way to define a goal in this theory is as optimising a function. This is relevant to our work since our agent will optimise for the shortest path in the traveling salesperson problem (TSP). Orseau et al. (2018) also state that an agent pursues goals.

The main motivation for this work stems from Andreas (2022), who asks if LMs can act as agents. He questions whether "LMs trained on text learn anything at all about the relationship between language and its use". He argues that "LMs are models of intentional communication in a specific, narrow sense. When performing next word prediction given a textual context, a LM can infer and represent properties of an agent likely to have produced that context". He also argues and shows that LMs infer and use abstract beliefs and goals. This is relevant because we want to see if an imitation learning agent can inherit the goals of an expert agent and its properties.

In the framework of Andreas (2022) can be understood the work by Li et al. (2023), who show that a LM can be trained to extend partial game transcripts of Othello in order to make a next legal move, which can be deemed a goal. Li et al. (2023) and Nanda et al. (2023) show that the LM also relies robustly on internal representations of the Othello board tiles and discs to make a next move.

Lovering et al. (2022) use AlphaZero, a reinforcement learning agent, and analyse it on the board game called Hex. They define the goal of Hex, and the goal of the agent, as to build a chain of pieces across the board, which wins the game. In Hex, certain templates have been recognised as useful to aid winning. Key part of learning how to play Hex is recognising when it is possible to connect groups of pieces together. The authors call these templates to be concepts within the game and are their primary focus of research. Our work focuses on goals as our problem setup is simpler than winning a game of Hex, thus we avoid using concepts. They also perform behavioural tests which evaluate specific types of out-of-distribution (OOD) generalisations, inline with our own experiments.

This dissertation explores the notion of *goal-directedness* (Shimi et al. (2021)) and what it means for a LM to act in a *goal-directed* manner. The definition of goal-directedness assumed here is that the model learns to perform a task and can exhibit adaptivity. Model adaptivity means the model will intentionally take actions to complete the task even when there is a systematic difference between the train and test set.

The presence of goal-directedness is tested in two ways. The first is to see if the model can generalise in specific, systematic ways that were not seen as part of the training data. This is along the insight of Shimi et al. (2021), who write that in literature there is an "apparent consensus that goal-directedness directly implies some level of generalisation." The second is to make the expert suboptimal at performing the task and see if the model is still able to achieve optimal behavior, i.e., outperform the expert and hence show a form of train-time adaptivity. Expert suboptimality is introduced by either random or systematic noise to its performance.

TSP poses a goal of optimally traversing all cities in the shortest distance covered. The aim is to show that, when trained on traversings generated by a predefined algorithm (the expert), the model learns to be goal-directed to solve TSP. This setup is called imitation learning. The decision to study TSP was inspired by Xing and Tu (2020) and Applegate et al. (2006).

How to pose TSP to the LM was influenced by Li et al. (2023). Our LM, like theirs, learns from pure sequence information and the model has no a priori knowledge of the game, the task and the rules. Moreover, as Andreas (2022) emphasizes, the LM is not trained to act in any environment or accomplish any goal beyond next-word prediction.

The bulk of the report conducts analysis on the five cities TSP using a 12 layer GPT-2 LM (Radford et al. (2019)) and a 2 layer GPT-2 LM called microGPT (Karpathy (2022)). Our results demonstrate the LM has the ability to find the optimal paths when presented new city coordinates which were not encountered during training. The LM can also find the optimal paths even when trained in presence of large amounts of systematic or random noise. Lastly, we attempt to show that goals are represented internally in the models activation using mechanistic interpretability.

Chapter 2

Method

2.1 TSP setup

In TSP, the salesperson aims to find the shortest possible route through a set of cities, visiting each city once and returning to the starting city. The choice to study the TSP is due to its unambiguously defined goal of finding this optimal route.

Cities are placed on a $2D$ grid of size 6×6 with integer coordinates. Each city is denoted by a capital letter in the English alphabet. An illustrative example in shown in Figure 2.1. In the Euclidean plane, the optimal TSP solution forms a simple polygon through all of the points, meaning it does not intersect itself and has no holes (Quintas and Supnick (1965)). In Figure 2.1 the optimal traversing stays the same up to a cyclic permutation. To maintain consistency and avoid introducing cyclic permutations, we choose to always start and end at city A.

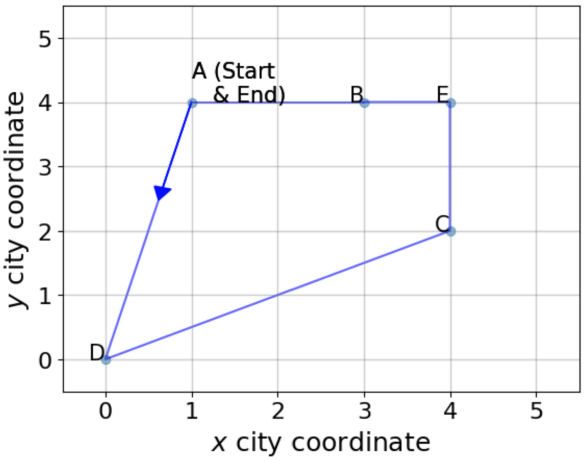
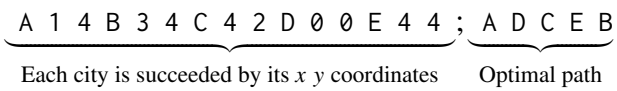


Fig. 2.1 TSP with five cities.

The TSP with the traversing is represented as a sequence of characters. For the example in Figure 2.1 the following sequence would be presented to the language model.



The main motivation for this sequence structure is to discriminate the goal from the rest of the sequence. The goal is represented as the optimal path, the last segment in the example above. The first segment defines the graph. The semicolon represents the delimiter between the graph and the salesperson traversing. We hypothesise that this aids learning because it explicitly denotes the end of graph input and the start of the traversing.

The LM is pre-trained on such sequences. For training and evaluation, we mask the tokens up to and including the starting city *A* in the last segment. So the model is trained on sequences such as in Figure 2.2.

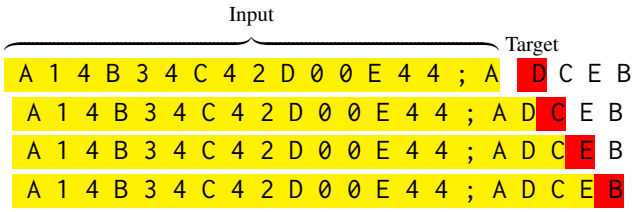


Fig. 2.2 Training sequences. Given an input (yellow) the model is queried to produce the targets (red).

2.2 Training Dataset

The main focus of this work was on the five cities TSP on a 6×6 grid. Because the city names are distinct letters, there are $36 \cdot 35 \cdot 34 \cdot 33 \cdot 32 = 45239040$ combinations of city placements, denoted as $45M$.

The datasets were generated using brute-force by selecting the shortest traversing. Since the salesperson starts from city A, there are $4! = 24$ possible traversing of the remaining four cities. Thus it was not too computationally expensive to check each traversing for all $45M$ examples. For a more sophisticated TSP solver and its attempt to be used, please see Appendix B.1.

For the example in Figure 2.1, an equivalently optimal path could have been going clockwise by first visiting city B instead of city D. This confounder may hamper model

learning, since a model would also have to learn when to produce an anticlockwise or clockwise path depending upon patterns in the input. To avoid this issue and ease learning, in our datasets anti-clockwise optimal paths were used.

2.3 Model

The LM used to run experiments was GPT-2. For detailed explanation on how the GPT-2 model works in general, please refer to Appendix A. Only the model adjustments are outlined here. Two different model sizes were used: GPT-2 and microGPT. All GPT-2 implementations used were from HF Canonical Model Maintainers (2022).

GPT-2 model has 12 decoder-only blocks and 12 attention heads. The embedding size used is $D = 780$ along with a vocabulary size of $|\mathcal{V}| = 13$. This gives a model with $\approx 88M$ trainable parameters. MicroGPT has 4 blocks and 4 attention heads. The embedding size is $D = 128$. With $|\mathcal{V}| = 13$, which gives $924k$ trainable parameters.

The choice of $|\mathcal{V}| = 13$ was because the alphabet of possible characters used were $\mathcal{V} = \{-100, 0, 1, 2, 3, 4, 5, ;, \texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}, \texttt{E}\}$. The character -100 represents the masking token. A simple tokeniser which maps this alphabet to the set of integers $\{0, 1, \ldots, 12\}$ was used for all experiments. This tokenisation avoids inductive biases of natural data present in the default GPT-2 tokeniser.

2.4 Model training

The model weights are randomly initialised for every run. The AdamW optimiser (Loshchilov and Hutter (2019)) was used with an initial learning rate η and weight decay λ, that applies to all layers except all bias and Layer-Norm weights.

The best training parameters for TSP were identified through a 2D grid sweep with $\eta \in \{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$ and $\lambda \in \{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$. For GPT-2, the optimal parameters were $\eta = 10^{-4}$ and $\lambda = 10^{-5}$, while for microGPT, they were $\eta = 10^{-3}$ and $\lambda = 10^{-4}$.

For GPT-2, a batch size of 2048 was used and evaluation was done every 800 training steps. For microGPT, a batch size of 65536 was used with evaluation performed every 25 training steps. Greedy decoding was used for both training and evaluation.

The model computes the Cross Entropy (CE) loss for each token prediction in Figure 2.2. The CE loss for a single example is the mean of the per-token CE loss. The overall CE loss $\mathcal{L}$ is the mean of the CE loss across all examples, representing the negative log-likelihood (NLL). Please see Appendix A for a detailed explanation of CE.

Two metrics were used for evaluating model performance: *Exact Match* ($\mathbb{EM}$) and *Per Token Accuracy* (PTA). For a single example, exact match is a binary metric indicating whether the model made all correct predictions, with a value of 1 if all predictions are correct and 0 otherwise (see Figure 2.3). For a set of $|\mathcal{E}|$ examples (where $\mathcal{E}$ denotes the evaluation set), exact match $\mathbb{EM}$ is the average per-example exact match, calculated as $\mathbb{EM} = \frac{1}{|\mathcal{E}|} \sum_{i=1}^{|\mathcal{E}|} \mathbb{EM}_i$. Exact match is the metric used to indicate the model's success in predicting the optimal TSP traversal, where suboptimal traversals result in $\mathbb{EM}_i = 0$ and optimal traversals yield $\mathbb{EM}_i = 1$. PTA indicates how many correct token predictions the model made overall.

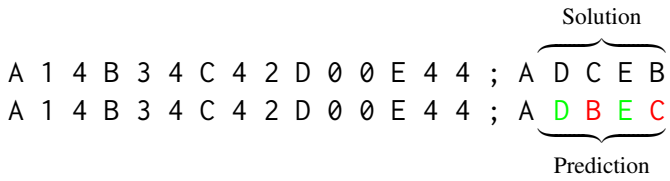


Fig. 2.3 The model made two correct and two incorrect predictions. Therefore, exact match $\mathbb{EM} = 0$ since all correct predictions were not made. The Per Token Accuracy (PTA) is $\frac{2}{4} = 0.5$ due to two correct predictions.

Chapter 3

Results and Discussion

The outline of this chapter is as follows. Section 3.1 defines the baseline models which show that learning is occurring and are used as a reference point. Sections 3.2, 3.3, 3.4, 3.5 and 3.6 contain various behavioural tests in order to demonstrate goal-directedness of our LMs.

Section 3.2 demonstrates the model's ability to generalise to a previously unseen coordinate at $(0,0)$, called origin. Extending upon this, Sections 3.3 and 3.4 are more difficult generalisation problems with more unseen coordinates at the board centre. They demonstrate that the LM is able to complete goals out of distribution. Specifically. the idea in these experimental setups is to ensure the test data is OOD in distinct ways while preserving the correctness of the training set. Sections 3.5 and 3.6 conduct experiments where noise is added to the training sets by perturbing the optimal paths. They demonstrate the model's ability to remain goal-directed by optimally solving tasks even when trained on noisy datasets. The noise is introduced as random and systematic noise in the optimal salesman trajectory in Sections 3.5 and 3.6, respectively.

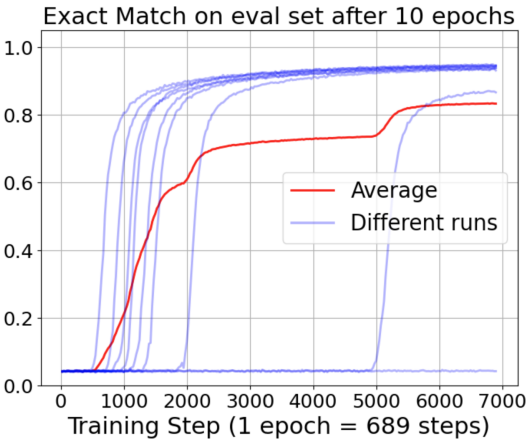
3.1 Baseline Models

The first task is to establish the performance of a baseline model. It will be used as a reference point for the performances of models in other experiments. These results show model's performance when trained on all available data (except the evaluation and test sets).

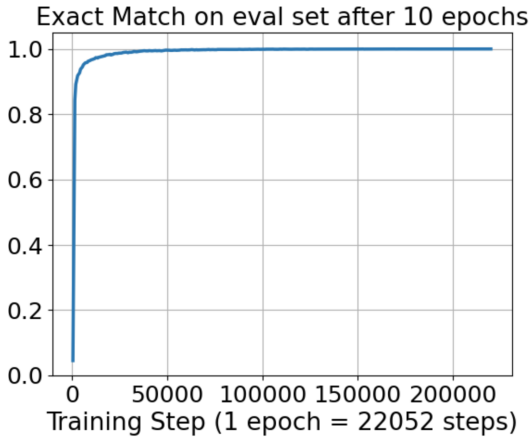
From the set of all combinations, an evaluation set with $27k$ examples and a test set with $50k$ examples were extracted uniformly at random without replacement. The performance of GPT-2 and microGPT are shown in Figures 3.1 and 3.2. Consistently, the GPT-2 model learned to solve the task. Therefore, only one average training run is presented for it.

Most of the time microGPT learned to solve the tasks, but there was one interesting outlier run in Figures 3.1 and 3.2 which did not learn to solve TSP. Thus we cannot claim

that microGPT models will learn to solve the TSP every time. The best achieved exact match for microGPT and GPT-2 are $\mathbb{EM} = 0.945$ and $\mathbb{EM} = 1$, respectively. These values will be used as a reference measure to see how well the subsequent models performed.

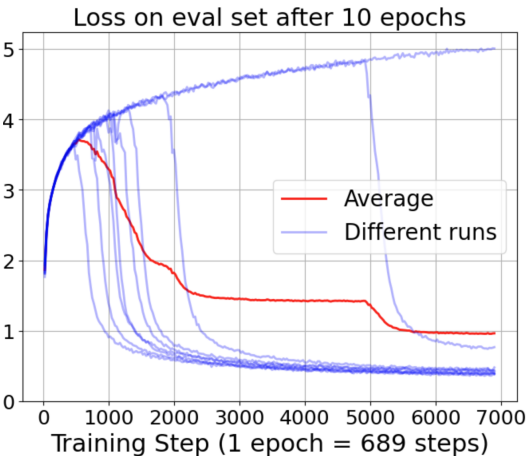


(a) microGPT (934k trainable parameters) multiple training runs

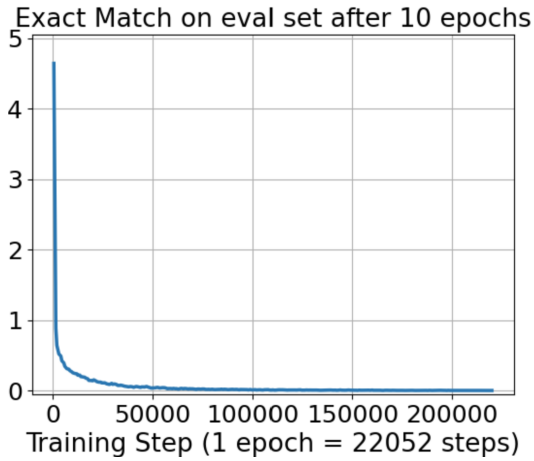


(b) GPT-2 (88M parameters)

Fig. 3.1 Exact match after training for 10 epochs



(a) microGPT (934k trainable parameters) multiple training runs



(b) GPT-2 (88M parameters)

Fig. 3.2 Negative Log-Likelihood after training for 10 epochs

3.2 Generalisation to Origin

On the 6×6 board the origin is defined as coordinate $(0,0)$. The total number of configurations where a city is present at the origin is $5 \cdot 35 \cdot 34 \cdot 33 \cdot 32 = 6283200$, denoted as $6.3M$.

Any of the five cities can be placed at the origin and the remaining four cities can be placed on the remaining 35 places in $35 \cdot 34 \cdot 33 \cdot 32$ combinations. Denote this set of configurations as $\mathcal{O}$, with cardinality $|\mathcal{O}| = 6.3M$.

TSP problems where a city is present at the origin we call *type-O* problems. One such example is shown in Figure 2.1. If an example was to be sampled uniformly, the probability that it will be type-O is $\mathbb{P}(\textit{type-O}) = \frac{5 \cdot 35 \cdot 34 \cdot 33 \cdot 32}{36 \cdot 35 \cdot 34 \cdot 33 \cdot 32} = \frac{5}{36} = 0.1389\,(13.89\%)$. The number of examples which are not type-O is total number of examples minus type-O examples: $(36-5) \cdot 35 \cdot 34 \cdot 33 \cdot 32 = 38955840$, denoted as $39M$. This set was shuffled once and used as the training set, denoted with $\mathcal{T}$. The set of coordinates present in train set $\mathcal{T}$ is shown in Figure 3.3.

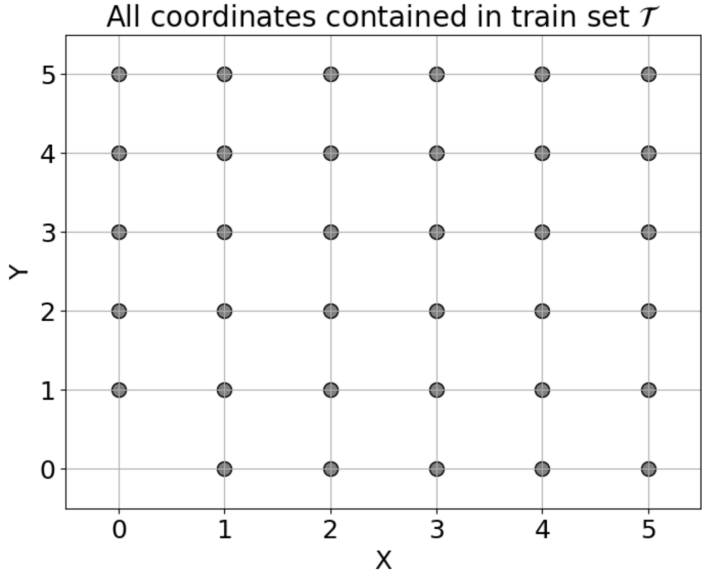


Fig. 3.3 Visualisation of coordinates the model sees when trained on set $\mathcal{T}$. It sees no type-O examples

The evaluation set $\mathcal{E}$ used has $27k$ examples, by subsampling randomly from set $\mathcal{O}$. Thus $\mathcal{E} \subset \mathcal{O}$ and $|\mathcal{E}| = 27k$. The network is trained on set $\mathcal{T}$ but evaluated on set $\mathcal{E}$ containing only type-O examples.

Additional training sets were created to vary the number of examples that had a city at the origin. An ensemble of sets $\mathcal{o}_1, \mathcal{o}_2, \ldots, \mathcal{o}_{10}$ was taken from set $\mathcal{O}$ such that $\mathcal{o}_1 \subset \mathcal{o}_2 \subset \cdots \subset \mathcal{o}_{10} \subset \mathcal{O} \setminus \mathcal{E}$. Their cardinalities are shown in Table 3.1.

By combining the training set $\mathcal{T}$ with sets $\mathcal{o}_1, \ldots, \mathcal{o}_{10}$ an ensemble of new training sets $\mathcal{T}_1, \ldots, \mathcal{T}_{10}$ is created. Let $i \in \{1, 2, \ldots, 10\}$. To generate set $\mathcal{T}_i$ we take the set $\mathcal{T}$, remove $|\mathcal{o}_i|$ examples from $\mathcal{T}$ and instead replace them with set $\mathcal{o}_i$. Thus the cardinality of set $\mathcal{T}_i$ is the same as $\mathcal{T}$, i.e. $|\mathcal{T}_i| = |\mathcal{T}| = 39M$. But now set $\mathcal{T}_i$ contains a certain fraction of type-O

Table 3.1 Cardinality of Sets

Set	$\mathcal{T}$	$\mathcal{O}$	$\mathcal{o}_1$	$\mathcal{o}_2$	$\mathcal{o}_3$	$\mathcal{o}_4$
Cardinality	39M	6.3M	1k	2k	5k	10k
Set	$\mathcal{o}_5$	$\mathcal{o}_6$	$\mathcal{o}_7$	$\mathcal{o}_8$	$\mathcal{o}_9$	$\mathcal{o}_{10}$
Cardinality	20k	50k	100k	200k	500k	1M

Table 3.2 Training sets with various numbers of type-O examples

Train set	$\mathcal{T}$	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$	$\mathcal{T}_4$	
#examples of type-O	0	1k	2k	5k	10k	
Fraction (%)	0	0.0026	0.0051	0.0128	0.0257	
Train set	$\mathcal{T}_5$	$\mathcal{T}_6$	$\mathcal{T}_7$	$\mathcal{T}_8$	$\mathcal{T}_9$	$\mathcal{T}_{10}$
#examples of type-O	20k	50k	100k	200k	500k	1M
Fraction (%)	0.0513	0.1284	0.2567	0.5134	1.2835	2.5670

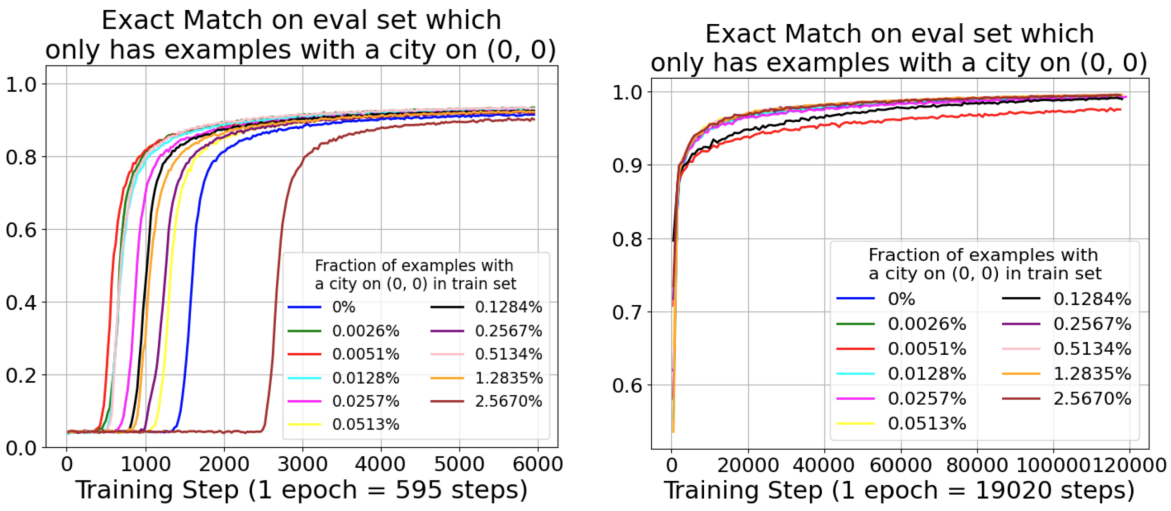
examples. These training sets are shown in Table 3.2. Note that the fraction of type-O examples present in each set $\mathcal{T}_i$ is less than $\mathbb{P}(type-O) = 13.89\%$. So each set $\mathcal{T}_i$ contains fewer type-O examples than the baseline set from Section 3.1.

During training, examples from set $\mathcal{T}_i$ are shown to the model in batches. Hence we ordered the examples in set $\mathcal{T}_i$ so that each batch has the same number of type-O examples present in it (up to ±1 type-O example). The results of training microGPT and GPT-2 are shown in Figure 3.4.

For the GPT-2 model exact match converges to $\mathbb{EM} = 1$, while for microGPT it converges to $\mathbb{EM} = 0.94$, both meeting the baseline performance from Section 3.1. Even in the case when the model is trained on $\mathcal{T}$, when the origin is never shown, the models reach these performances. Hence the model is able to generalise and accomplish the goal of traversing the given problems in shortest path.

The model produces optimal solutions to problems containing the origin. Note that even though the model never sees the origin coordinate (0, 0) during training, it does see the character '0'. This indicates the model's capability to understand what it means to bind two zeros together, in order to create a new coordinate, namely (0, 0).

In Figure 3.4a it is evident that presenting the model with a million type-O examples (brown) did not improve learning compared to when no type-O examples were shown (blue). Thus, it is not necessarily the case that increasing the number of type-O examples will yield better results.



(a) microGPT (b) GPT-2

Fig. 3.4 Model generalises to solve examples in eval set which *only* has examples with a city at the origin (type-O examples).

3.3 Generalisation to More Excluded Points

A more difficult generalisation task is to omit a bigger segment of the 6×6 grid in the training set than the one from Section 3.2. In this section the excluded points are defined as coordinates $(2,2)$ and $(2,3)$. A setup comparable to Section 3.2 of the training and evaluation sets is used here.

The number of points on the 6×6 grid excluding these points is 34. Hence the number of examples that do not contain the two points is $34 \cdot 33 \cdot 32 \cdot 31 \cdot 30 = 33390720$, denoted as $33.4M$. This is used as the training set called $\mathcal{T}$, with $|\mathcal{T}| = 33.4M$. Visualisation of these points is shown in Figure 3.5.

Examples which do contain a city on at least one of $(2,2)$ or $(2,3)$ we call *type-C* examples (C as in 'centre'). There are $45M - 33.4M = 11848320$ such examples, denoted with $11.9M$. This set is denoted with C and has cardinality $|C| = 11.9M$. Examples which contain exactly one city on either $(2,2)$ or $(2,3)$ are called *type-*C_1 examples. Examples which contain cities on both $(2,2)$ and $(2,3)$ at the same time are called *type-*C_2 examples. In Figures 3.6a and 3.6b are shown instances of type-C_1 and type-C_2 examples, respectively.

Now we count the number of type-C_2 examples. There are $5 \cdot 4$ ways to place two cities on $(2,2)$, $(2,3)$ and the remaining three cities can be placed in $34 \cdot 33 \cdot 32$ ways. Hence there are $5 \cdot 4 \cdot 34 \cdot 33 \cdot 32 = 718080$ such examples, denoted as $700k$. This set is denoted with C_2. So the number of *type-*C_1 examples is $11.9M - 700k \approx 11.2M$. This set is denoted with C_1.

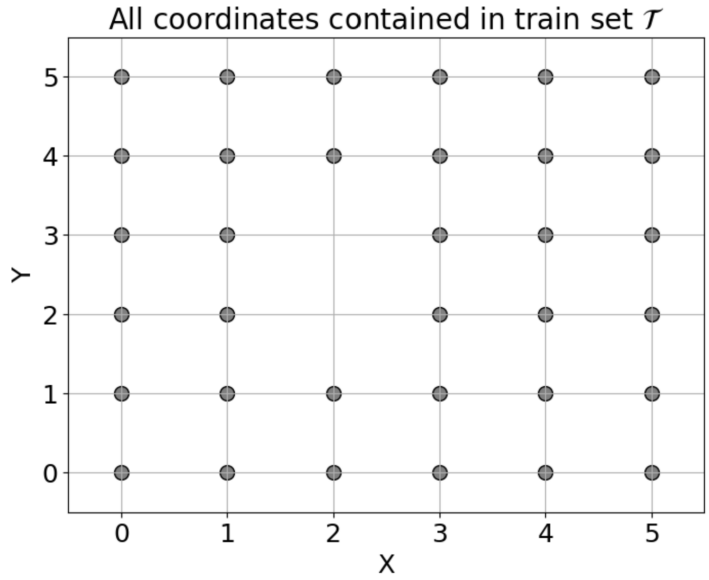
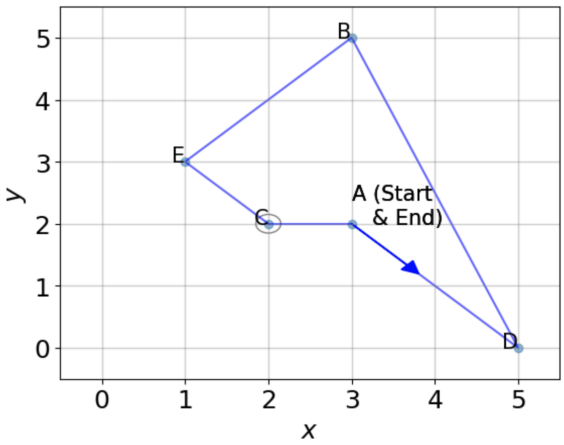
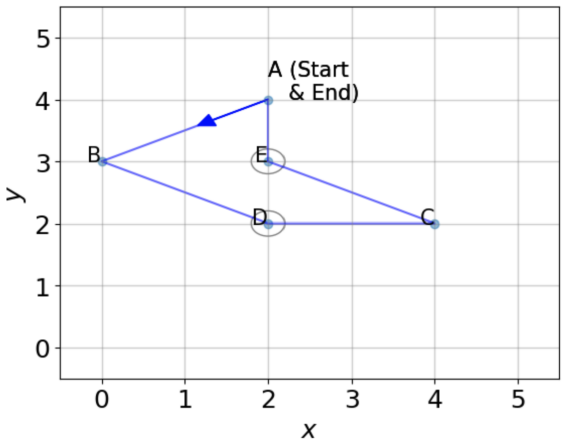


Fig. 3.5 Visualisation of coordinates the model sees when trained on set $\mathcal{T}$. It sees no type-C examples.

Together the union of sets C_1 and C_2 encapsulate all type-C examples $C = C_1 \cup C_2$, and the set $\mathcal{T} \cup C_1 \cup C_2$ has all $45M$ examples.



(a) Type-C_1 example. Contains exactly one city on $(2,2)$ or $(2,3)$.



(b) Type-C_2 example. Contains cities on $(2,2)$ and $(2,3)$ at the same time.

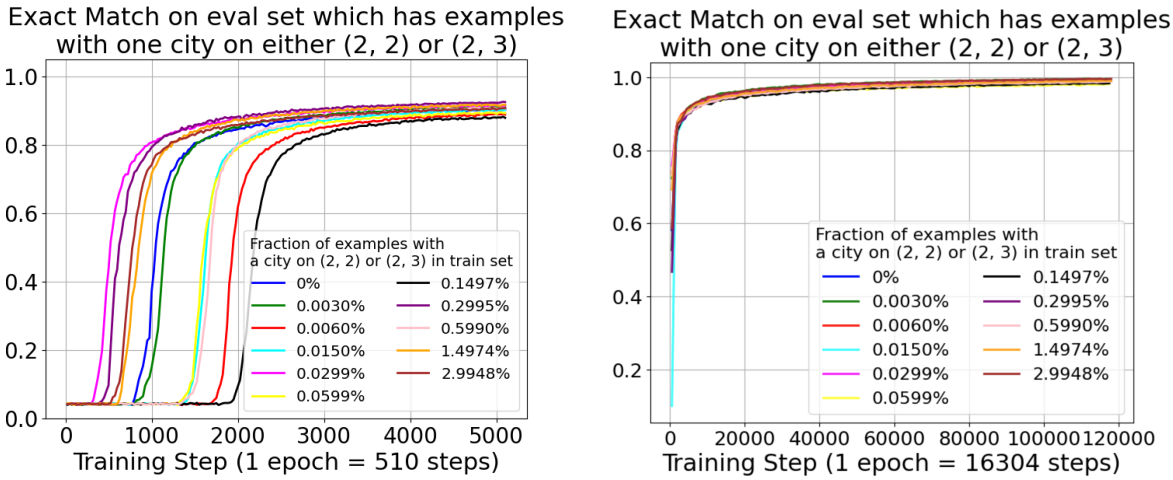
Fig. 3.6 Type-C examples have either one or two cities on $(2,2)$ and $(2,3)$

Two different evaluation sets $\mathcal{E}_1$ and $\mathcal{E}_2$ of cardinalities $|\mathcal{E}_1| = |\mathcal{E}_2| = 27k$ are created. $\mathcal{E}_1$ is created by randomly subsampling $27k$ examples from C_1 without replacement, so $\mathcal{E}_1 \subset C_1$. In the same way $\mathcal{E}_2$ is created by subsampling C_2, thus $\mathcal{E}_2 \subset C_2$. In words, evaluation set $\mathcal{E}_1$ contains *only* type-C_1 examples and evaluation set $\mathcal{E}_2$ contains *only* type-C_2 examples.

Table 3.3 Training sets with various numbers of type-C_1 examples

Train set	$\mathcal{T}$	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$	$\mathcal{T}_4$	
#examples of type-C_1	0	1k	2k	5k	10k	
Fraction (%)	0	0.0030	0.0060	0.0150	0.0299	
Train set	$\mathcal{T}_5$	$\mathcal{T}_6$	$\mathcal{T}_7$	$\mathcal{T}_8$	$\mathcal{T}_9$	$\mathcal{T}_{10}$
#examples of type-C_1	20k	50k	100k	200k	500k	1M
Fraction (%)	0.0599	0.1497	0.2995	0.5990	1.4974	2.9948

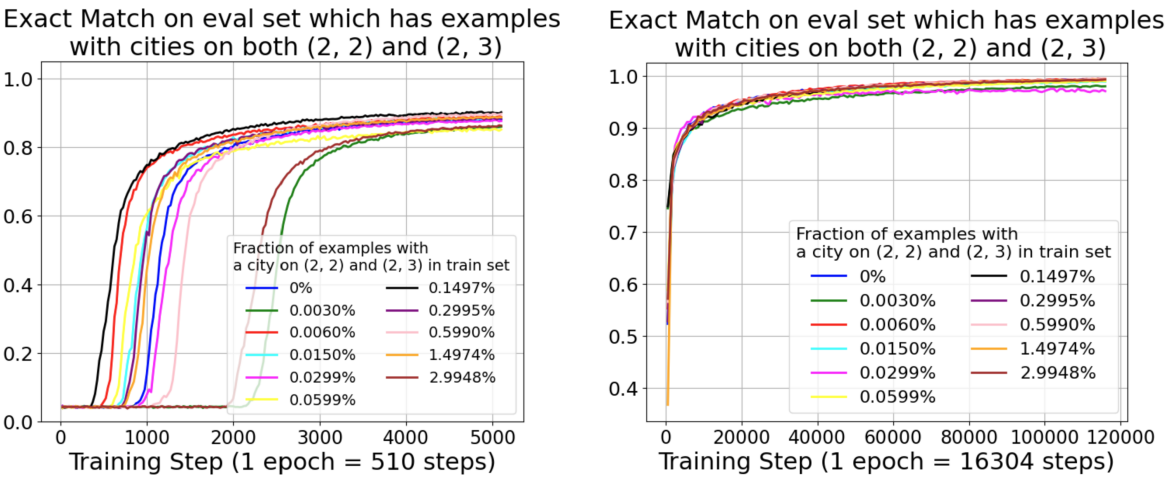
Now the aim is to generate additional training sets $\mathcal{T}_1, \ldots, \mathcal{T}_{10}$ from the train set $\mathcal{T}$. This is done in exactly the same way how they were generated in Table 3.2 from Section 3.2, but this time $\mathscr{o}_1 \subset \mathscr{o}_2 \subset \cdots \subset \mathscr{o}_{10} \subset C_1 \setminus \mathcal{E}_1$ (as opposed to $\subset \mathcal{O} \setminus \mathcal{E}$ in Section 3.2). The training sets are shown in Table 3.3. Each set $\mathcal{T}_i$ has the same cardinality as $\mathcal{T}$, i.e., $|\mathcal{T}_i| = |\mathcal{T}| = 33.4M$, but now each $\mathcal{T}_i$ contains a certain fraction of type-C_1 examples and contains no type-C_2 examples. The results of training on sets from Table 3.3 with evalutation sets $\mathcal{E}_1$ and $\mathcal{E}_2$ are shown in Figures 3.7 and 3.8, respectively.



(a) microGPT (b) GPT-2

Fig. 3.7 Model generalises to solve problems on set $\mathcal{E}_1$

For GPT-2 the model's performance does seem to converge to $\mathbb{EM} = 1$ in Figures 3.7b and 3.8b, irrespective of the fraction of type-C_1 examples shown during training. Even when trained on set $\mathcal{T}$ where the excluded points $(2,2)$, $(2,3)$ are never shown, GPT-2 manages to optimally solve examples when either one or two cities are present at the centre. This tells us the model is goal-directed.



(a) microGPT (b) GPT-2

Fig. 3.8 Model generalises to solve problems on set $\mathcal{E}_2$

As for microGPT's performance, in Figure 3.7a the final values fall in $\mathbb{EM} \in [0.85, 0.90]$ and in Figure 3.8a the final values are in the range $\mathbb{EM} \in [0.878, 0.925]$. Both performances are very good, given that they are just slightly lower than the $\mathbb{EM} = 0.945$ baseline microGPT model from Section 3.1. This indicates the model is goal-directed. In Figures 3.7a and 3.8a it is interesting that the zero percent curve does better than some of the other curves which contain some relevant training examples.

Similar to Section 3.2 when the model is trained on set $\mathcal{T}$, it never gets to see coordinates (2, 2) and (2, 3), but it does see 2 and 3 individually. This information seems sufficient for the model to have the capability to understand what it means to bind these characters together in order to form new coordinates, specifically (2, 2) and (2, 3).

3.4 Generalisation to Centre

Driven by the success of Section 3.3, the attempt in this section is to omit a bigger segment of the 6×6 board from the training to make the generalisation task harder. The coordinates $(2,2)$, $(2,3)$, $(3,2)$ and $(3,3)$ form what we call the *centre*.

The are 32 coordinates on the 6×6 board without the 2×2 centre. Those are shown in Figure 3.9. Consequently, there are $32 \cdot 31 \cdot 30 \cdot 29 \cdot 28 = 24165120$ ($24M$) examples that do not have a city in the centre. This set is denoted with $\mathcal{T}$ with $|\mathcal{T}| = 24M$. Set $\mathcal{T}$ is the only training set used.

Please note the experimental setup here is simpler than those from Sections 3.2 and 3.3, where it was noted that adding examples of type-O and type-C respectively, did not enhance

the network's performance. As a result, the decision was made to train the network on set $\mathcal{T}$ without ever exposing it to the centre. Therefore, all training plots in this section are equivalent to the 0% curves from Sections 3.2 and 3.3.

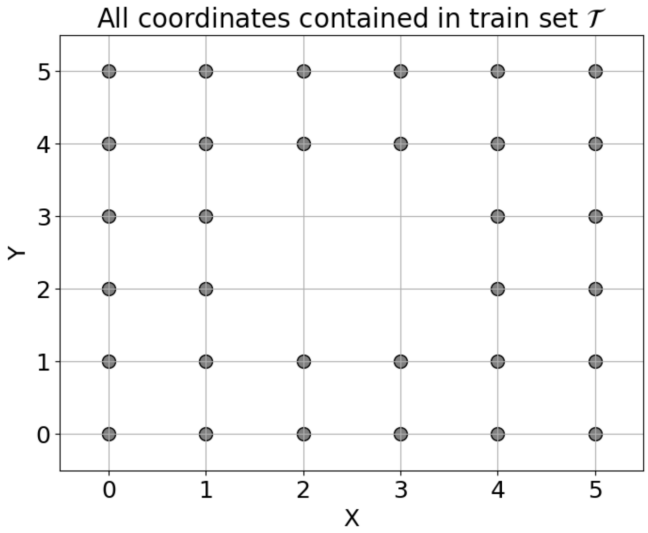
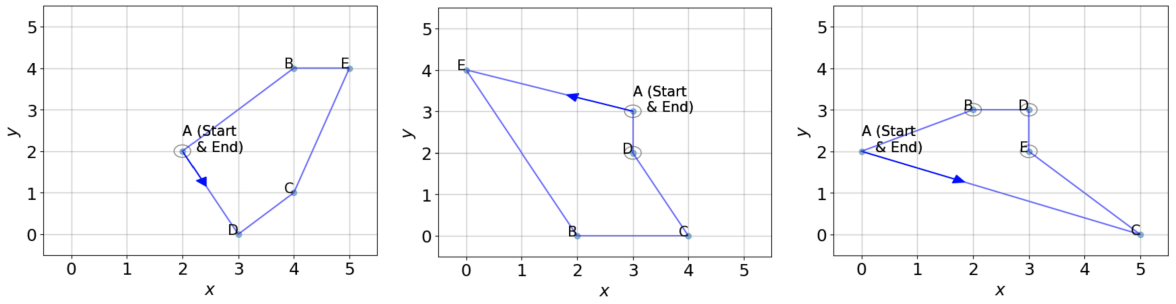


Fig. 3.9 Visualisation of coordinates the model sees when trained on set $\mathcal{T}$. Does not see any of the coordinates $(2,2)$, $(2,3)$, $(3,2)$ and $(3,3)$, which form the centre.

To generate the evaluation sets, first note that there are $45M - 24M = 20996920\,(21M)$ examples that have at least one city in the 2×2 centre. From this set we create three evaluation sets, denoted with $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}_3$ and all with $|\mathcal{E}_1| = |\mathcal{E}_2| = |\mathcal{E}_3| = 27k$. Sets $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}_3$ contain examples which have exactly one, two and three cities present in the centre, respectively. Figure 3.10 demonstrates such examples.



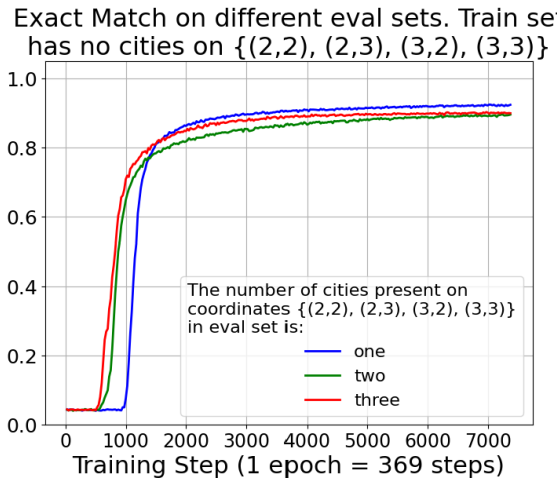
(a) Example with one city in the centre. Eval set $\mathcal{E}_1$ consists of such examples.

(b) Example with two cities in the centre. Eval set $\mathcal{E}_2$ consists of such examples.

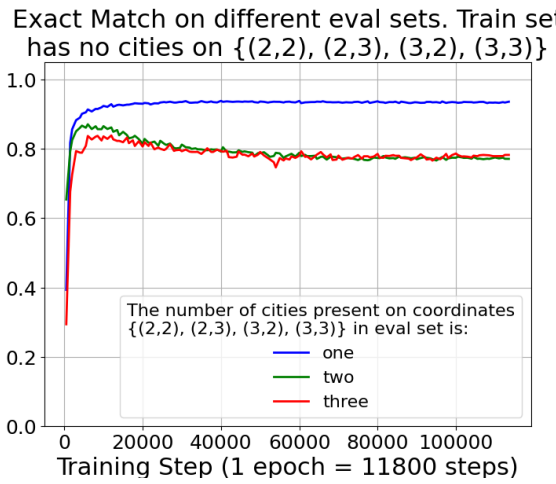
(c) Example with three cities in the centre. Eval set $\mathcal{E}_3$ consists of such examples.

Fig. 3.10 Different types of examples used in eval sets $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}_3$.

For the set $\mathcal{E}_1$, note that the city in the centre may be placed on any one of the four coordinates $(2,2)$, $(2,3)$, $(3,2)$ and $(3,3)$. Therefore, when constructing the set $\mathcal{E}_1$ we



(a) microGPT



(b) GPT-2

Fig. 3.11 Model generalises to solve examples when cities are present on 2x2 centre

ensure an equal distribution of examples, with $\frac{|\mathcal{E}_1|}{4}$ examples featuring a city at each of these coordinates. Analogously, since there can be $\binom{4}{2} = 6$ pairs of centre coordinates formed, the set $\mathcal{E}_2$ contains $\frac{|\mathcal{E}_2|}{6}$ of examples featuring each pair. Similarly for $\mathcal{E}_3$, it has $\frac{|\mathcal{E}|}{4}$ examples of each of the $\binom{4}{3} = 4$ triplets.

The results of training GPT-2 and microGPT on set $\mathcal{T}$ and evaluating on sets $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}_3$ are shown in Figure 3.11. It can be seen that in all cases the model's $\mathbb{EM}$ converges to a high value, thus indicating goal-directedness of the model. In Figure 3.11a on the eval set $\mathcal{E}_1$ the model reaches $\mathbb{EM}_1 = 0.923$ and $\mathbb{EM}_2 = 0.895$, $\mathbb{EM}_3 = 0.899$ for eval sets $\mathcal{E}_2$ and $\mathcal{E}_3$, respectively. In Figure 3.11b GPT-2 model reaches $\mathbb{EM}_1 = 0.935$, $\mathbb{EM}_2 = 0.77$ and $\mathbb{EM}_3 = 0.781$.

So with more unseen coordinates both microGPT and GPT-2 have worse end performance. This is because they only encounter one new coordinate during evaluation, as opposed to encountering two or three new coordinates. What is interesting is that GPT-2 slightly outperforms microGPT on $\mathcal{E}_1$, but is worse on sets $\mathcal{E}_2$ and $\mathcal{E}_3$.

When compared with Sections 3.2 and 3.4, the model's generalisation here is most difficult, due to

- the largest grid section being removed in the train set $\mathcal{T}$
- $|\mathcal{T}|$ being the smallest.

This explains why the models give slightly worse results than in previous sections.

3.5 Random Noise

In the previous sections the model was always trained on optimal data. In this and in Section 3.6 the aim is to train on noisy data and see how the model performs. Previous work shows that models are able to generalise when trained on noisy datasets (Lubana et al. (2023), Arpit et al. (2017), Arjovsky et al. (2020), Krueger et al. (2021)).

In our setup, noise is introduced by randomly swapping two of the last four cities in the optimal traversing.[1] The choice of which two cities to swap is uniform among the four cities. One such example in shown in Figure 3.12.

```
A 1 4 B 3 4 C 4 2 D 0 0 E 4 4 ; A D C E B
A 1 4 B 3 4 C 4 2 D 0 0 E 4 4 ; A C D E B
```

Fig. 3.12 Random Noise. Any two of the last four cities in the optimal path (the first sequence) are swapped, thus giving a suboptimal path (second sequence).

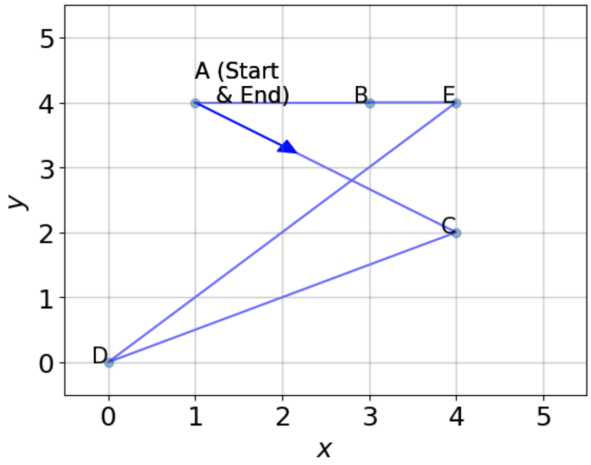


Fig. 3.13 Random noise is added to the optimal path from Figure 2.1. For this example, by swapping two cities the path self-intersects, making it suboptimal.

The base training set $\mathcal{T}$ with $|\mathcal{T}| = 45M$ examples and evaluation set $\mathcal{E}$ with $|\mathcal{E}| = 27k$ from Section 3.1 are used. An ensemble of training sets $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{10}$ is constructed from set $\mathcal{T}$. Let $i \in \{1, \ldots, 10\}$. Set $\mathcal{T}_i$ will contain $10\% \cdot i$ percent of suboptimal examples and $100\% - 10\% \cdot i$ percent of optimal examples. The sets are shown in Table 3.4. During

[1] Flipping two cities in an optimal path increases the path length (otherwise the original path was not optimal), making it suboptimal. Flipping may cause self-intersections, local or global loss of optimality, or other path issues.

training, we order the examples in set $\mathcal{T}_i$ so that each batch contains the same number of suboptimal examples (up to ±1). Likewise, each batch also contains the same number of optimal examples (up to ±1).

Table 3.4 Training Sets with various amount of suboptimal random noise paths

Set	$\mathcal{T}$	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$	$\mathcal{T}_4$	$\mathcal{T}_5$
Fraction of suboptimal ex.	0%	10%	20%	30%	40%	50%
Set		$\mathcal{T}_6$	$\mathcal{T}_7$	$\mathcal{T}_8$	$\mathcal{T}_9$	$\mathcal{T}_{10}$
Fraction of suboptimal ex.		60%	70%	80%	90%	100%

The results of training on sets from Table 3.4 are shown in Figures 3.14, 3.15 and 3.16.

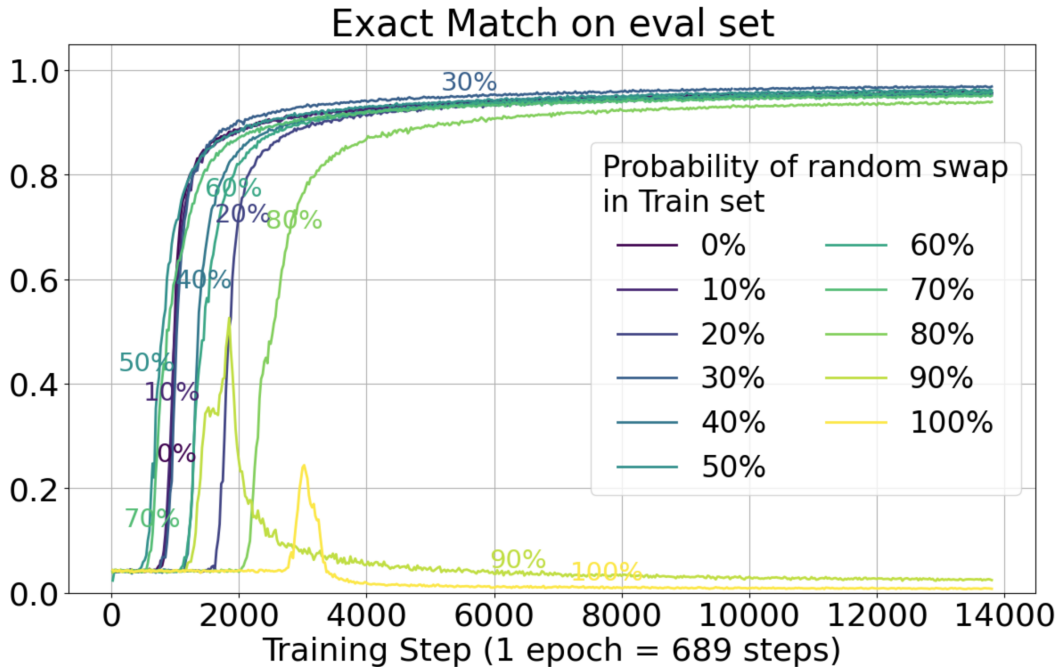


Fig. 3.14 microGPT. Training sets have various amount of noisy salesperson traversings called *probability of swap* in the legend. A noisy traversing has two cities swapped when compared with optimal traversing.

In Figures 3.14 and 3.16a it can be seen that the model is able to learn to solve the tasks optimally, even in the presence of large amounts of noise and thus be goal-directed. As a comparison, in classification problems there is evidence when a lot of label noise is present, the model can still learn the correct labels (Arpit et al. (2017)). Our setting is more tailored to goal-directedness. Image classification is a one-off task, whereas the TSP is sequential in nature, requiring the chaining together of previous outputs in the correct way, hence making mere memorization under noise more difficult.

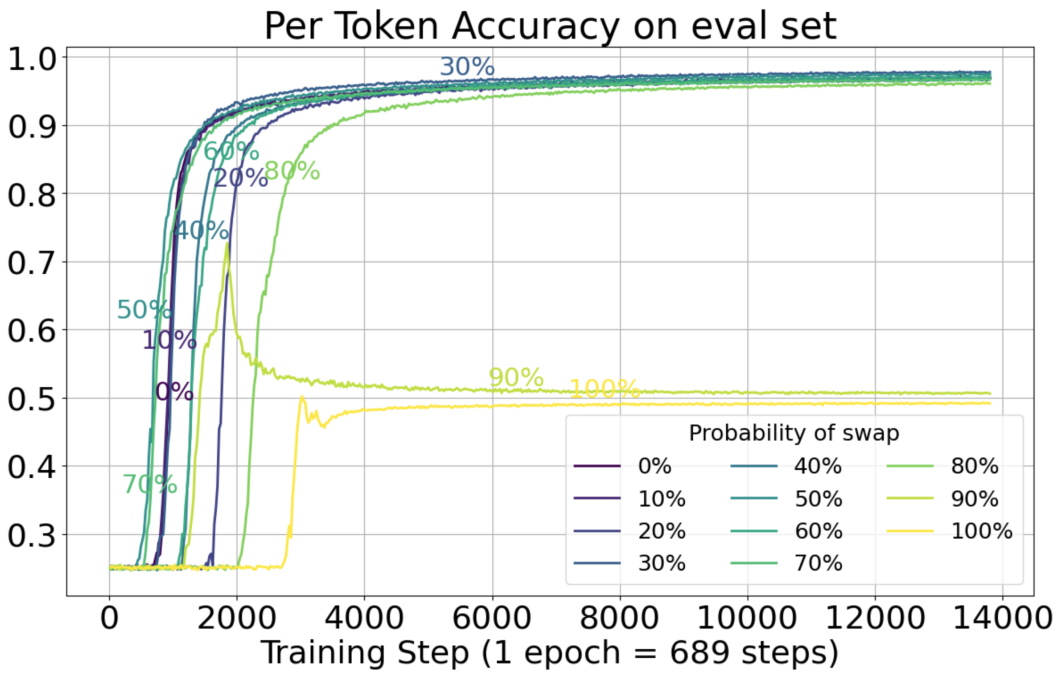
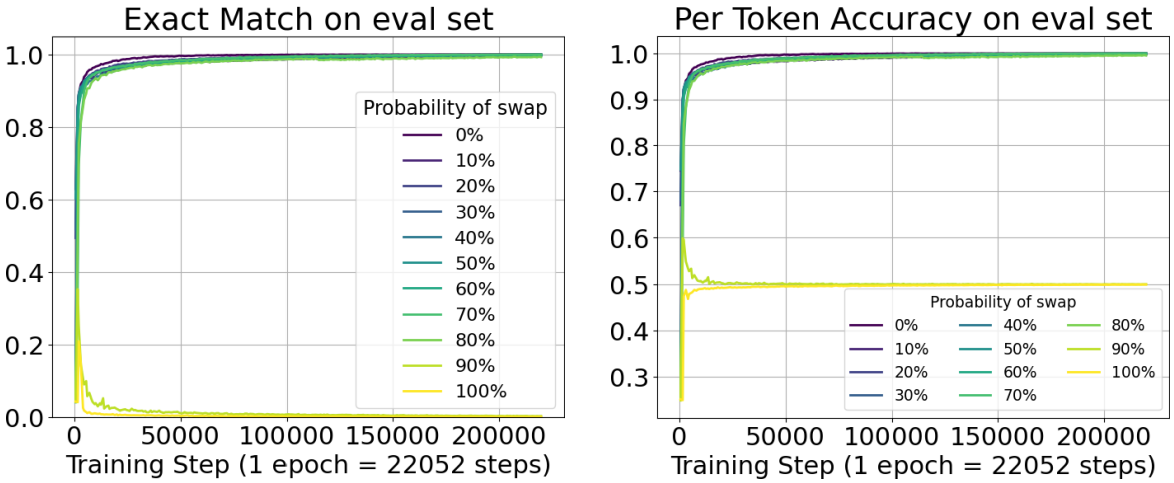


Fig. 3.15 microGPT. The setup corresponds to the one from figure 3.14



(a) Exact Match

(b) Per token accuracy

Fig. 3.16 GPT-2. Training sets have various amount of noisy salesperson traversings called *probability of swap* in the legend. A noisy traversing has two cities swapped when compared with optimal traversing.

In Figures 3.14 and 3.16a it can be seen that reducing capacity from GPT-2 to microGPT slows down training for both the optimal (0%) and noisy data. In general this slowing down is more extreme on noisy datasets (Arpit et al. (2017)), but this was not necessarily always the case in our experiments (e.g. the 0% curve rises later than 50% or 70% curves in Figure 3.14). Because of the slowing down effect, the microGPT was trained for 20 epochs and GPT-2 for 10 epochs.

In Figure 3.16b, the per-token accuracy (PTA) lines converge either to 0.5 or 1. They converge to 1 when the model learns to optimally solve all examples. When there is too much noise present in the training dataset, the PTA converges to 0.5. Looking back at Figure 2.2, the model needs to make four predictions per example, but in Figure 3.12 only two letters get swapped. The other two letters remain at the correct positions. So half of the cities are in their correct locations for a noisy example. If the whole dataset has noisy examples ($\mathcal{T}_{10}$, 100% curve), then half of the cities will be at the right place. This explains why the model gets 0.5 of the per-token guesses correct.

3.6 Systematic Noise

In this section systematic noise is introduced to the optimal salesperson path, achieved by always swapping cities B and C. An example is shown in Figure 3.17. Such examples are called *swapped examples*, as opposed to optimal examples.

```
A 1 4 B 3 4 C 4 2 D 0 0 E 4 4 ; A D C E B
A 1 4 B 3 4 C 4 2 D 0 0 E 4 4 ; A D B E C
```

Fig. 3.17 Systematic Noise. Cities B and C are swapped in the optimal path (first sequence) are swapped, thus giving a suboptimal path (second sequence).

Recall the baseline training set $\mathcal{T}$ from Section 3.1 with $|\mathcal{T}| = 45M$ and evaluation set $\mathcal{E}$ with $|\mathcal{E}| = 27k$. They contain the optimal paths and will be used to train and evaluate the model. Additional training sets will be generated from set $\mathcal{T}$.

An additional evaluation set $\mathcal{E}_{BC}$ is made from the set $\mathcal{E}$ by swapping cities B and C in the optimal paths. $\mathcal{E}_{BC}$ contains all the same examples $|\mathcal{E}| = |\mathcal{E}_{BC}|$, with the same input (up to and including letter `A` after the delimiter token, i.e., `;`) but with the suboptimal path. So during evaluation when the model makes a prediction, we can compare the generated path with respect to the optimal and swapped paths at the same time.

Table 3.5 Training sets with various fractions of swapped examples present. All sets have the same size.

Train set	$\mathcal{T}$	$\mathcal{T}_{10}$	$\mathcal{T}_{20}$	$\mathcal{T}_{30}$	$\mathcal{T}_{40}$	$\mathcal{T}_{42}$	$\mathcal{T}_{44}$	$\mathcal{T}_{46}$
Fraction (%)	0	10	20	30	40	42	44	46
Train set	$\mathcal{T}_{47}$	$\mathcal{T}_{48}$	$\mathcal{T}_{49}$	$\mathcal{T}_{50}$	$\mathcal{T}_{51}$	$\mathcal{T}_{52}$	$\mathcal{T}_{53}$	$\mathcal{T}_{54}$
Fraction (%)	47	48	49	50	51	52	53	54
Train set	$\mathcal{T}_{56}$	$\mathcal{T}_{58}$	$\mathcal{T}_{60}$	$\mathcal{T}_{70}$	$\mathcal{T}_{80}$	$\mathcal{T}_{90}$	$\mathcal{T}_{100}$	
Fraction (%)	56	58	60	70	80	90	100	

An ensemble of training sets of same size with various amounts of swapped examples was generated and is shown in Table 3.5. Each set has a fraction $f\%$ of swapped examples and $(100-f)\%$ of optimal examples. The way each set was constructed is by ensuring every successive group of a 100 examples taken from $\mathcal{T}$ has f swapped examples and $100-f$ optimal examples, which are evenly distributed in the group. This ensures each training batch observes roughly the same number of swapped and optimal examples.

Results of training on sets from Table 3.5 with evaluation on set $\mathcal{E}$ are shown in Figures 3.18, 3.20 and with evaluation on set $\mathcal{E}_{BC}$ are shown in Figures 3.19, 3.21.

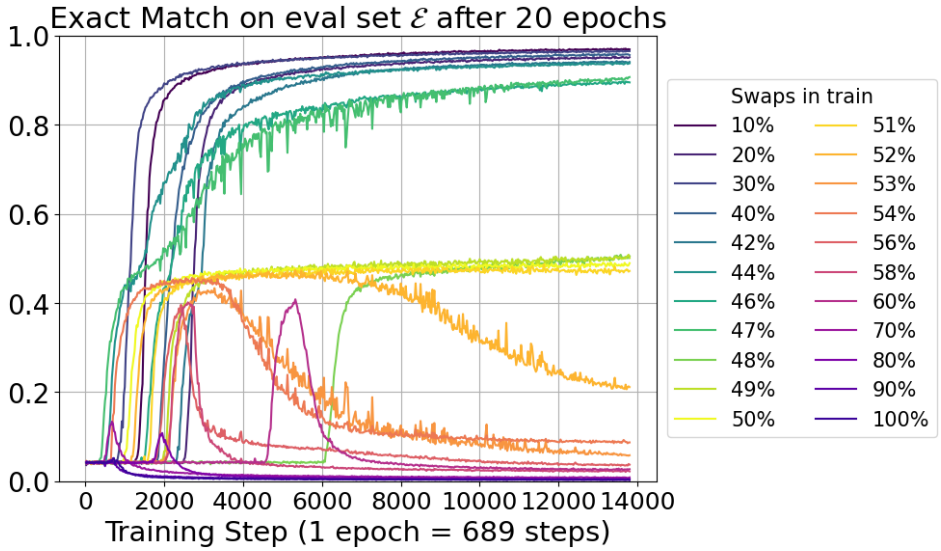


Fig. 3.18 microGPT. Exact match with respect to optimal paths in eval set $\mathcal{E}$. The model is trained on datasets which contain various amounts of examples with a noisy salesperson traversing.

In Figures 3.18 and 3.20 it can be seen that below 50% of swaps, the curves converge to their respective baseline $\mathbb{EM}$ performance, which is with respect to the optimal path.

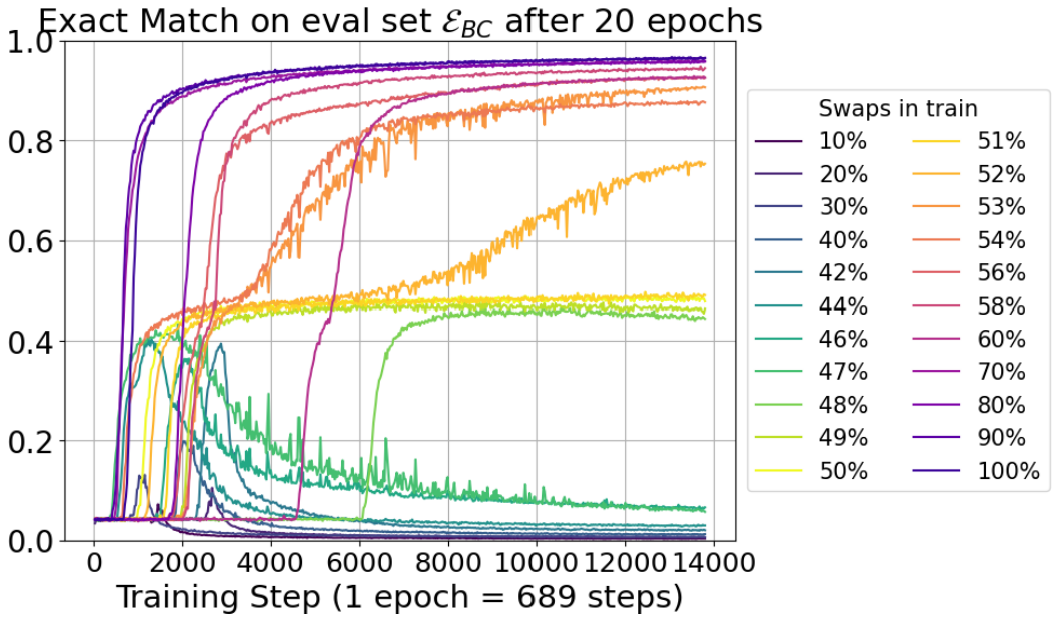


Fig. 3.19 microGPT. Exact match with respect to swapped paths in eval set $\mathcal{E}_{BC}$. The model is trained on datasets which contain various amounts of examples with a noisy salesperson traversing.

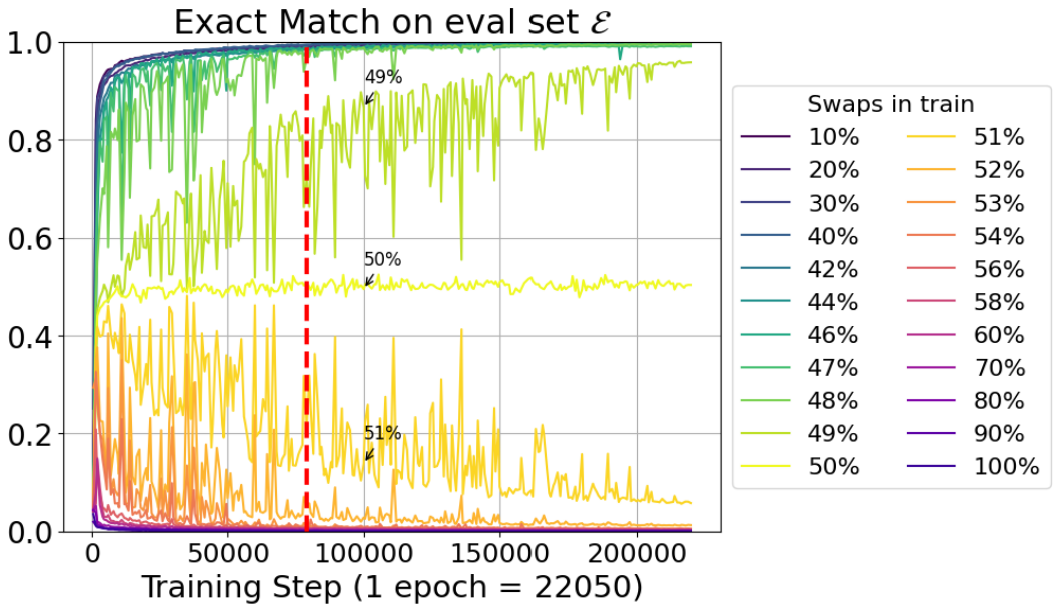


Fig. 3.20 GPT-2. Exact with respect to optimal paths in eval set $\mathcal{E}$. The model is trained on datasets which contain various amounts of examples with a noisy salesperson traversing. Red dotted line is used in Figure 3.22.

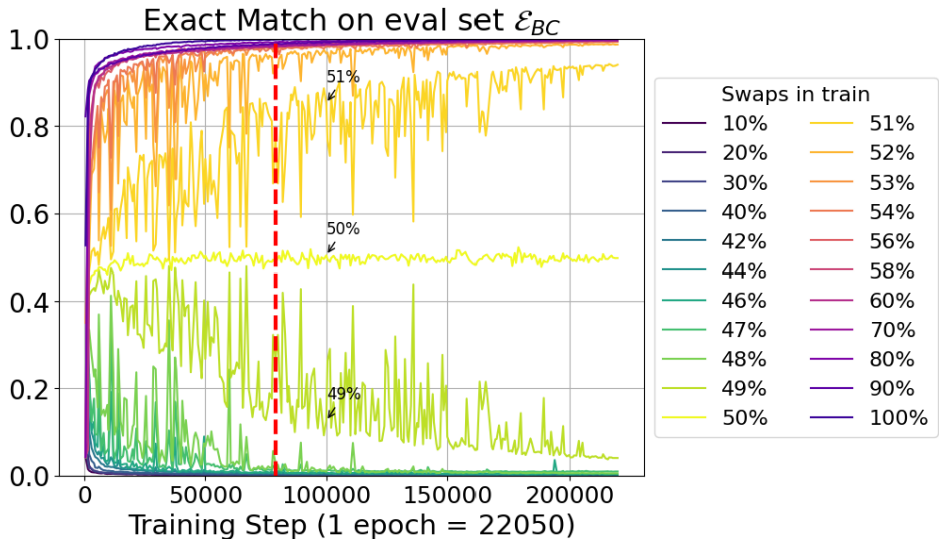


Fig. 3.21 GPT-2. Exact match with respect to swapped paths in eval set $\mathcal{E}_{BC}$. The model is trained on datasets which contain various amounts of examples with a noisy salesperson traversing. Red dotted line is used in Figure 3.22.

However, below 50% the curves tend to $\mathbb{EM}$ of 0. When trained on set $\mathcal{T}_{50}$, $\mathbb{EM}$ stays around 0.5.

On the contrary, during the same training runs, we also evaluate on set $\mathcal{E}_{BC}$ and $\mathbb{EM}$ is measured with respect to swapped solutions. This is shown in Figures 3.19 and 3.21. These $\mathbb{EM}$s converge nearly to 1 when there are more than 50% of swaps, while the $\mathbb{EM}$s converge to 0 when there are less than 50%. When trained on set $\mathcal{T}_{50}$, $\mathbb{EM}$ again converges to 0.5, just like in Figures 3.18 and 3.20.

After training has finished, it is as if Figures 3.18 and 3.19 are mirrored about $\mathbb{EM} = 0.5$. Likewise for Figures 3.20 and 3.21.

The model converges to make predictions of whichever goals occur more in the training set. Why this happens we hypothesise could be explained either by:

1. memorisation with greedy decoding or

2. model acts in an agentic manner, meaning it governs its choice of actions and considers its beliefs of which goals the train data expert possesses (see Chapter 1). Consequently, the actions the model takes may or may not lead it to the actual expert goal.

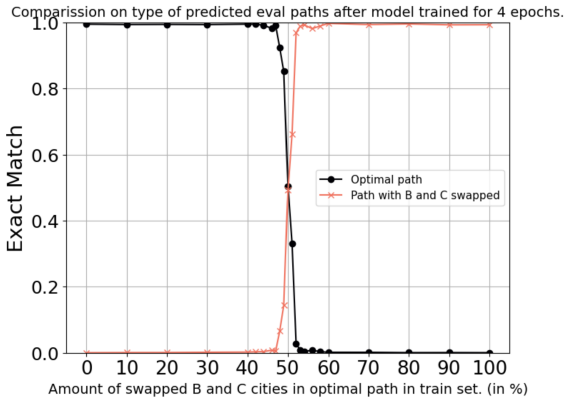
Before elaborating 1., note that each example in a set $\mathcal{T}_i$ occurs only once in that set, with either an optimal or noisy path. So each input is assigned either an optimal path or a noisy path. A counterpart in classification problems would be that each training image is assigned exactly one label, either 1 or 0, there is no re-randomisation of the labels. And a certain

image will always appear with, e.g., label 1 during multi-epoch training. With this setup the network could potentially memorise the label for each given training input.

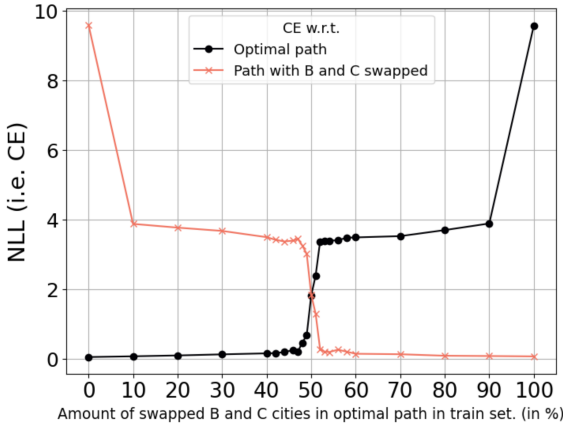
Memorisation in LMs is a phenomenon where the model retains specific training data information. The model would strictly 'memorise' each training example after observing it one or multiple times during training. In classification problems, when the model sees a certain training input, it has memorised the corresponding label, which it then outputs.

Memorisation is distinct from generalisation, where the model learns underlying patterns in the training set to generate new content. Memorisation may occur because of overfitting. The model learns the training data too well, including any noise, instead of capturing the underlying patterns. Note that memorisation happens on the train set. The interesting thing is that even when a model has memorised, it can have great generalisation performance (Arpit et al. (2017)).

Returning to our problem, it is possible that the model is memorising the training set. But good generalisation combined with greedy decoding yields great performance on the evaluation sets, as seen in Figures 3.18, 3.19, 3.20 and 3.21. Now we investigate if that is the case. First we plot the cross sections of Figures 3.20 and 3.21 after 3.58 training epochs, shown in Figure 3.22a. The corresponding loss also after 3.58 epochs is shown in Figure 3.22b. This number of epochs was chosen in order to give the model a fair chance to memorise.



(a) Exact match with respect to 1) optimal path (black) and 2) swapped path (orange)



(b) NLL with respect to 1) optimal path (black) and 2) swapped path (orange)

Fig. 3.22 GPT-2 performance after 3.58 training epochs on evaluation sets $\mathcal{E}$ (black) and $\mathcal{E}_{BC}$ (orange).

Figure 3.22a is explained by greedy decoding. Once the number of swapped examples in the training set surpasses 50%, the model switches to output swapped paths because it decodes in a greedy manner. This explains why the plots resemble step functions.

Nonetheless, Figure 3.22 is what is interesting. Note that NLL is independent of the type of decoder used, yet the step like curves are visible around 50%. This is not typical and what was expected to be seen is a smooth increase (decrease) for the black (orange) curve. Note that for $f\%$ of swaps, if the gap between the black and orange is larger, it indicates the network is more confident that the lower curve solution is superior. To investigate further Figure 3.22b, we conduct an analysis of the NLL on a per-token basis across the batch.

Recall from Section 2.4 how the 'overall' loss $\mathcal{L}$ on the eval set (shown in Figure 3.22b and everywhere up to here) is computed as the mean CE loss across both the *target length* (T) dimension ($T = 4$, the four targets from Figure 2.2) and the eval set size dimension. Now the *per-token loss* is introduced, which is the mean CE loss only across the eval set size dimension. The target length dimension remains. It is computed for each token in red from Figure 2.2 across the evaluation set. The losses are shown in Figure 3.23, whose top plot is the same as Figure 3.22b (the 'overall' loss). Also note the top plot in Figure 3.23 is such that it is the mean of the four per-token loss plots below it.

The x-axis represents the amount of swaps present each training set. Let there be $x\% = 100\% \cdot x$ of swaps, where x is the probability of a swapped example occurring in the train set $\mathcal{T}_{x\%}$.

Let us explain what is the purple curve. It represents the training agent behaviour, i.e. the performance of the expert who generated the underlying dataset. This is called the loss of the training expert. The NLL $\mathcal{L}$ of the training expert can be computed as $\mathcal{L}_{expert} = \mathbb{P}(\texttt{swapped path}) \cdot \mathcal{L}_{BC} + \mathbb{P}(\texttt{optimal path}) \cdot \mathcal{L}_{opt.} = x \cdot \mathcal{L}_{BC} + (1-x) \cdot \mathcal{L}_{opt.}$. If the model had learned the exact distribution of the training dataset $\mathcal{T}_x$, its loss would be the purple curve. But *the model actually learns the orange or black curve, whichever is lower for a given* $\mathcal{T}_x$.

We can see that when making the token 0 prediction (e.g. the first row, red target in Figure 2.2), the gap between the black and orange curves in Figure 3.23 is tiny for any $\mathcal{T}_x$, meaning the model is not very confident which path (optimal or swapped) to choose. But since the model uses greedy decoding, it will always decode from whichever curve is lower.

Once the model generates token 0, it appends token 0 to the input sequence and uses that as the new input (second row, Figure 2.2). The model moves on to look at the NLL of token 1 in Figure 3.23. But notice that the gap between black and orange curves increases. This means the *model has now become more confident which path to follow*. After generating token 1, it moves on to generate token 2 and again the gap increases and the model is more confident which path to follow. The last token 3 generation is very confident which path to choose, because by then only one more unvisited city remains.

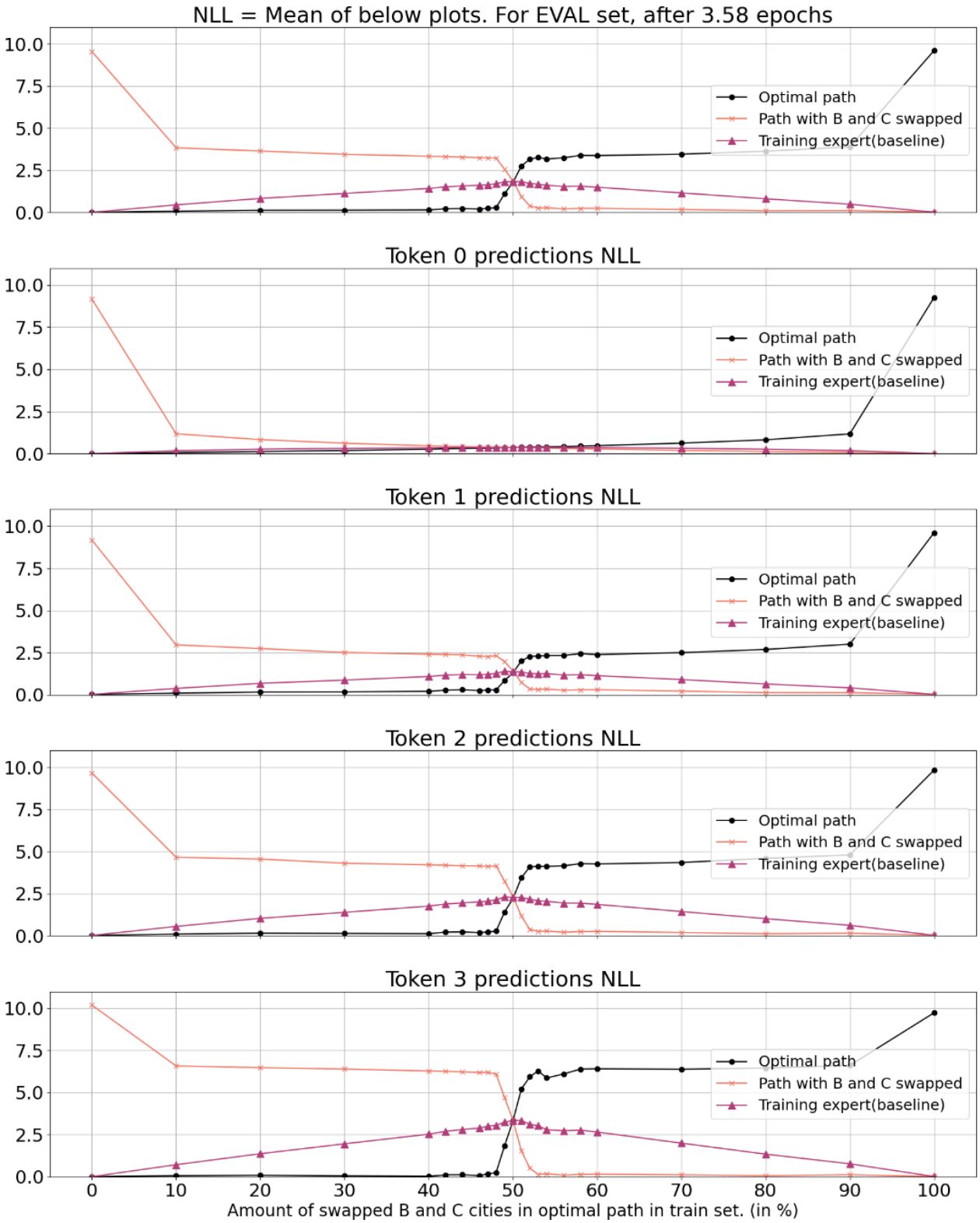


Fig. 3.23 Per-Token NLL for GPT-2 after 3.58 epochs on the evaluation set. Token 0 position correspond to the first row, red target position in Figure 2.2. Similarly for tokens 1, 2, 3. The x-axis represents the percentage of training examples with swapped paths. So each value on the x-axis corresponds to a different training set configuration used for training the model.

The conclusion is that *the model generalises overconfidently on the evaluation set*, even though the true underlying trend is the purple curve. Note that Figure 3.23 was on the eval set $\mathcal{E}$ for the black curves and on $\mathcal{E}_{BC}$ for the orange curves. A similar analysis is now conducted but on a set of the first $27k$ examples from the training set $\mathcal{T}$. Those $27k$ examples have been observed four times during training, thus giving the model a chance to memorise. The NLL plots are shown in Figure 3.24.

The key here in the interpretation is NLL of token 0's prediction. In Figure 3.23, on the evaluation set, the model was not confident which path to take. If the model has memorised the train set, then the model would be very confident which path to take for token 0 plot in Figure 3.24. This would manifest as a large gap between the black and orange curves. But this is not the case. Instead the model is not confident at what it should predict for token 0, just like it was not confident in Figure 3.23. The token 0 plot for the eval set in Figure 3.23 looks the same as the token 0 plot for the train set in Figure 3.24. Though not a definitive proof, this indicates that the train set has not been memorised.

To determine if the model intentionally chooses between the swapped or optimal path as suggested in point 2. and to explain the results in Figures 3.23 and 3.24, we turn our attention to mechanistic interpretability.

3.7 Mechanistic Interpretability

Mechanistic interpretability (Elhage et al. (2021), Nanda et al. (2023)) tries to understand how individual or groups of components of a model (such as those in Figure A.1) work together to generate outputs. It looks at the *internal* numbers, operations, mechanisms and structures of the model in order to understand how the model processes information, makes decisions and generates outputs.

We examine the activation patterns and feature representations within each of the four blocks in microGPT, trained on the $\mathcal{T}_{46\%}$ dataset. Smaller models like microGPT are easier to interpret than bigger models such GPT-2, due to their reduced number of parameters and layers. By analysing activation patterns, we aim to identify any recurring patterns in neuron outputs, thereby gaining insights into the model's functional components. Feature representation analysis tries to discern how various input data features are represented internally in the model. Mechanistic interpretability is best used as a tool when coupled with behavioural experiments to validate the consistency between the model's representations and its observed behaviour.

The objective is to examine whether we can interpret the representations to see if hypothesis 2. is true from Section 3.6.

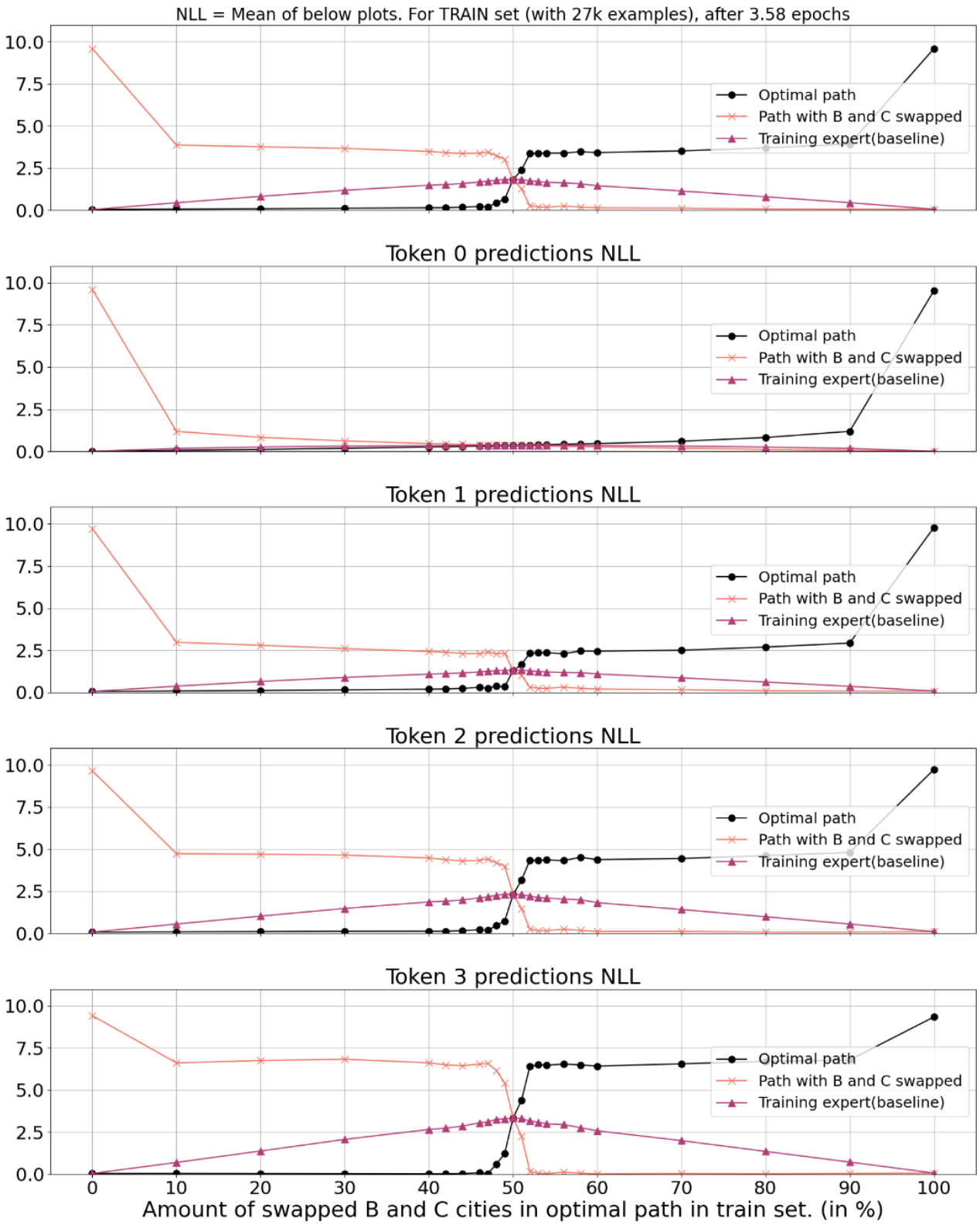


Fig. 3.24 Per Token NLL for GPT-2 after 3.58 epochs on $27k$ train set examples. Token 0 position correspond to the first row, red target position in Figure 2.2. Similarly for tokens 1, 2, 3. The x-axis represents the percentage of training examples with swapped paths. So each value on the x-axis corresponds to a different training set configuration used for training the model.

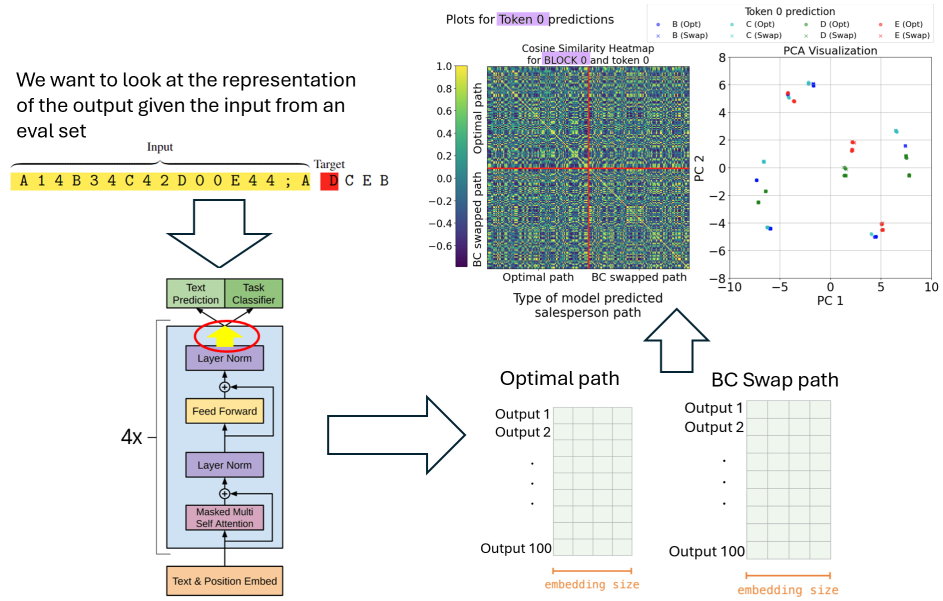


Fig. 3.25 Pipeline used for Mechanistic Interpretability. The activation embeddings of token A from each microGPT block are extracted for 200 examples. From these, a cosine similarity matrix is constructed on which PCA is done. Bottom right figure by Alammar (2019). Bottom left figure by Radford and Narasimhan (2018).

The approach is illustrated in Figure 3.25. Given an example input (top left), taken from the evaluation dataset $\mathcal{E}$, various neurons in the model activate. First, we store the activations of the final input token (letter A) at the end of each block (yellow arrow, bottom left). For the last token, the output of each block will be a 1D array with a length equal to the token embedding size, $D = 128$, which we wish to extract. Since microGPT has four blocks, this results in four 1D arrays, each representing the state of token A at the end of a block. Activation retrieval and storage are facilitated by *forward hooks*, which capture activation arrays during the forward pass (bottom left).

The above was a per-example description. Now we opt to store the activations of token A for a 100 examples whose whole generated output sequence is optimal and for a 100 examples whose generated output sequence is the swapped path (bottom left). This is feasible because the model consistently outputs either the optimal or swapped path for each example. In essence, we store the activations of token A for 100 examples where the model generated the optimal path and another 100 examples where it produced the swapped path. In total, there are 200 activations stored, each of length $D = 128$. These can get stacked into one *activation matrix* of shape $200 \times D$.

A symmetric matrix of cosine similarities between each two examples can be generated. This *cosine similarity matrix* will be of shape 200×200 (top right). A bigger image of the

cosine similarity matrix is shown on the left of Figure 3.26. The i, j entry of this matrix will be the cosine similarity of the i-th and j-th row of the activation matrix. Cosine similarity of the two activation row vectors is defined as $\frac{a_i a_j^T}{||a_i|| \cdot ||a_j||} \in [-1, 1]$.

This cosine similarity matrix is partitioned into four quadrants. The top left (bottom right) quadrant corresponds to cosine similarities between activations which resulted in the optimal (swapped) paths being generated. The other two quadrants correspond to cosine similarities between an activation which resulted in the optimal path and an activation which resulted in the swapped path.

The hypothesis is that there is a goal representation of following a swapped or optimal path stored in the cosine similarity matrix. Principal Component Analysis (PCA) can be conducted on the cosine similarity matrix to identify potential clusters of activations that may be interpretable.

Additionally, for each input sequence we store two pieces of information:

- The first generated token (e.g. the model generated `D` for the input in the first row of Figure 2.2)
- The type of the overall generated sequence. So after the model generated four tokens, the path is either an optimal ('o') or a path with B and C swapped ('x').

With this information, we can label each point in the PCA plot to indicate both the first token generated and the type of sequence produced by the model for a given input. Please see the caption in Figure 3.26. A set of such analysed cosine similarity matrices with the PCA plots are shown in Figures 3.26, 3.27, 3.28 and 3.29.

As the processing of data progresses through blocks in Figures 3.26, 3.27, 3.28 and 3.29, we can see the activations grouping up after each block to form four clusters. The last block activations are shown in 3.29 and four clusters are present, each of which has three colours. Note that every colour is present in three clusters. It is as if the model learned to *not* put a colour in one of the four clusters.

The same pipeline from Figure 3.25 can be used for prediction of tokens 1, 2 and 3. Described here is for token 1 prediction. Once the model generates token 0, this token feeds back into the input sequence and is used as a new input. For example, the model generates `D` in the second row in Figure 2.2. Now we can again repeat the pipeline in Figure 3.25 with this new input sequence to get the PCA visualisation of the token 1 prediction. A similar procedure follows for generating tokens 2 and 3 visualisations.

The plots for the last token prediction (last row of Figure 2.2) are shown in Figures 3.30, 3.31, 3.32 and 3.33.

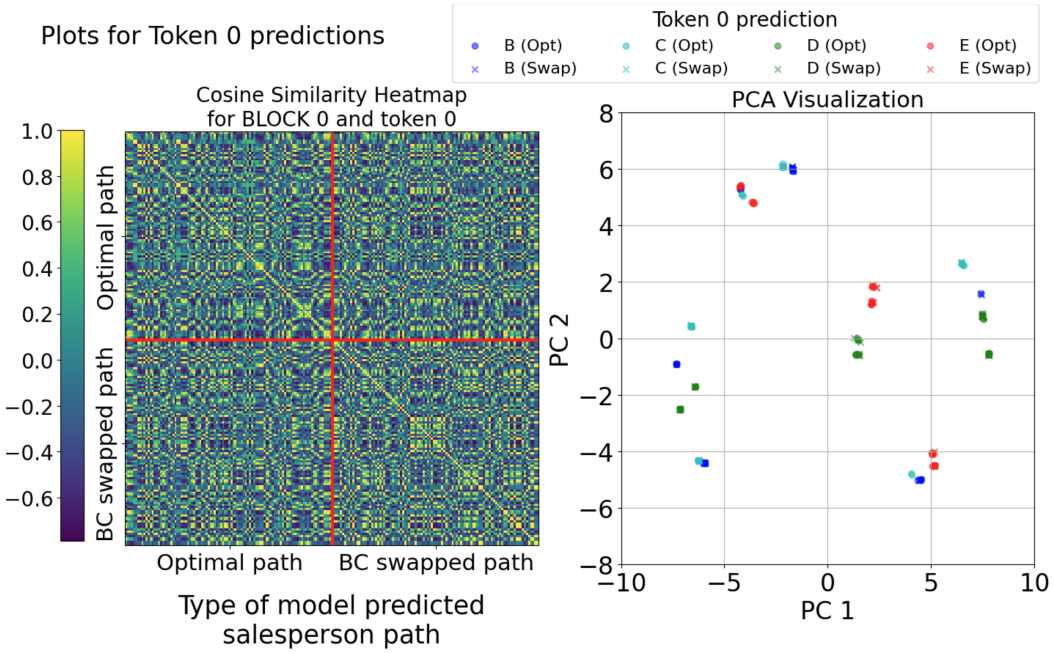


Fig. 3.26 Letter 0, Block 0. The left Figure shows the cosine similarly matrix between every two input activations. There are 200 different inputs, hence this matrix is of shape 200×200. On this matrix PCA is performed. It reduces the 200×200 matrix to a 200×2 PCA matrix. Thus each of the 200 activations is now represented by two coordinates, shown in the right figure. The legend indicates for each activation what letter did it generate and what is overall type of path the model generated.

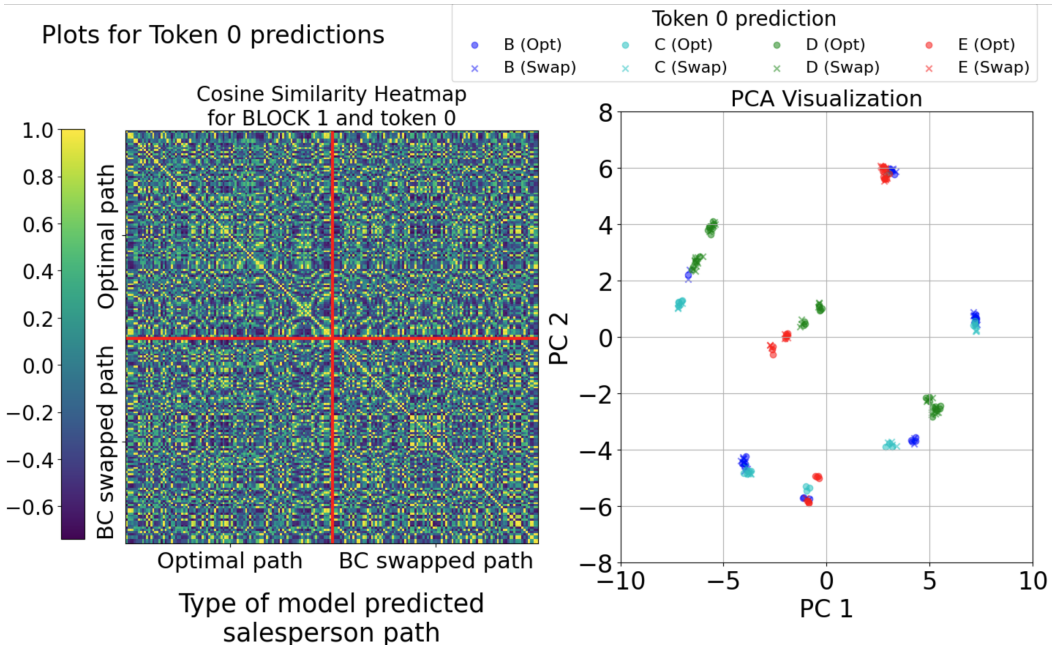


Fig. 3.27 Letter 0, Block 1

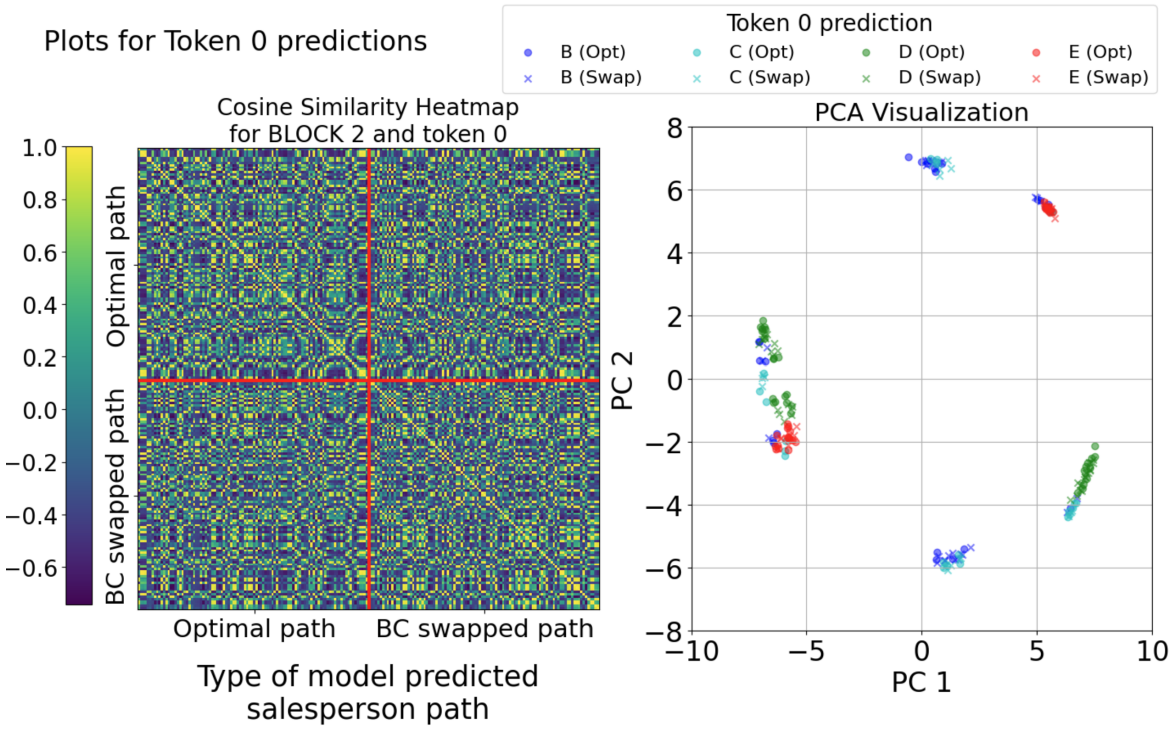


Fig. 3.28 Letter 0, Block 2

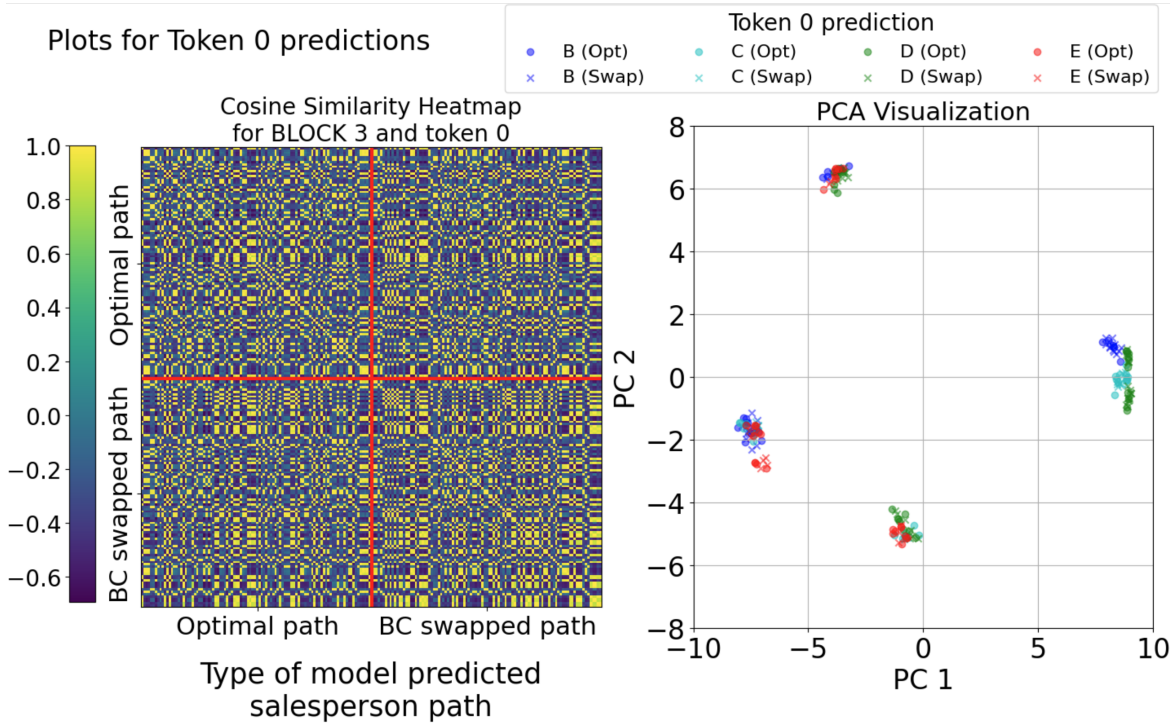


Fig. 3.29 Letter 0, Block 3

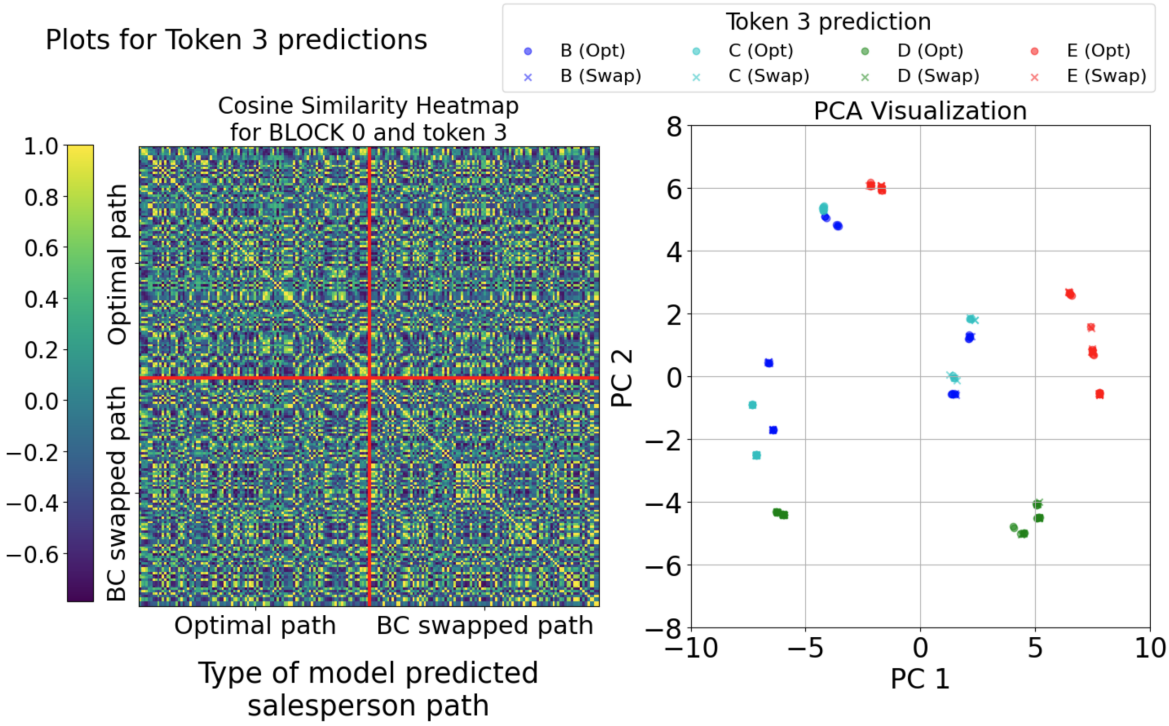


Fig. 3.30 Letter 3, Block 0

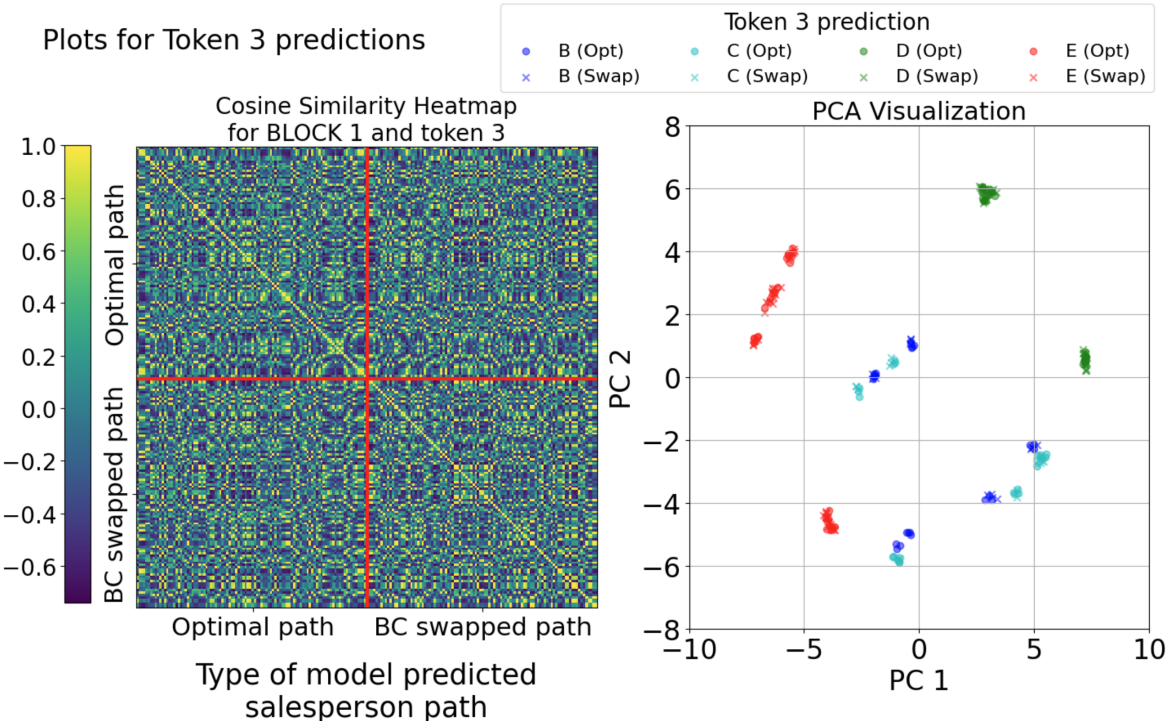


Fig. 3.31 Letter 3, Block 1

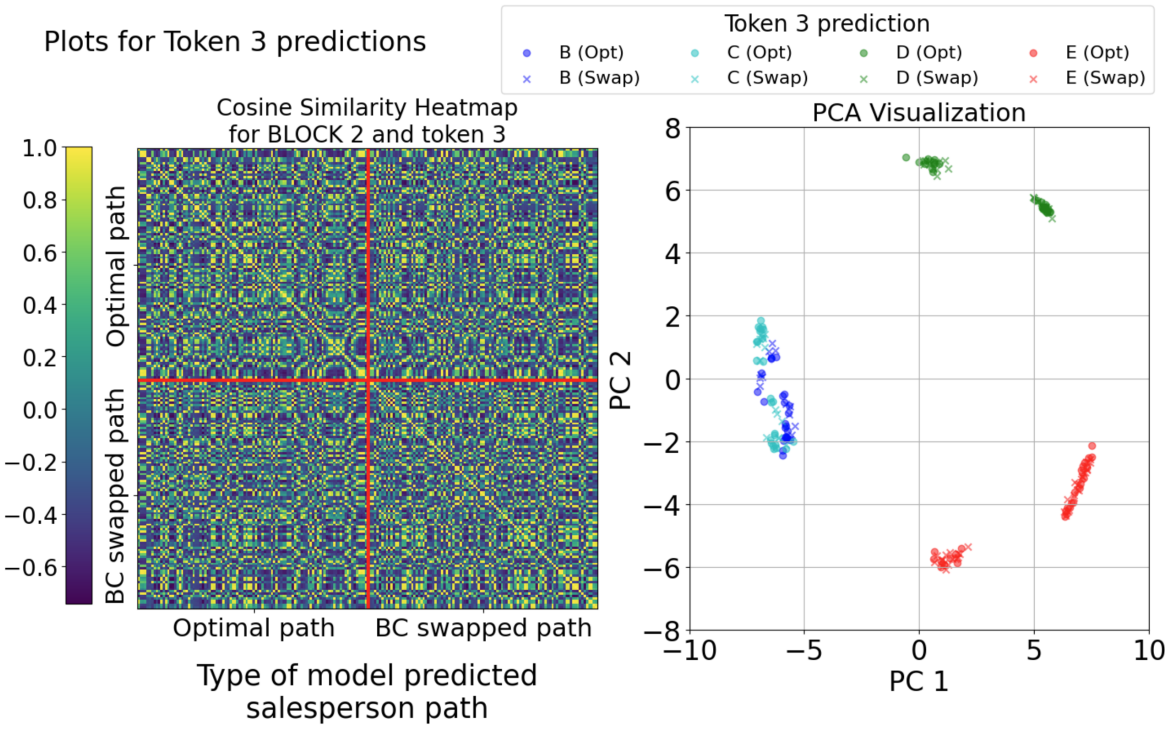


Fig. 3.32 Letter 3, Block 2

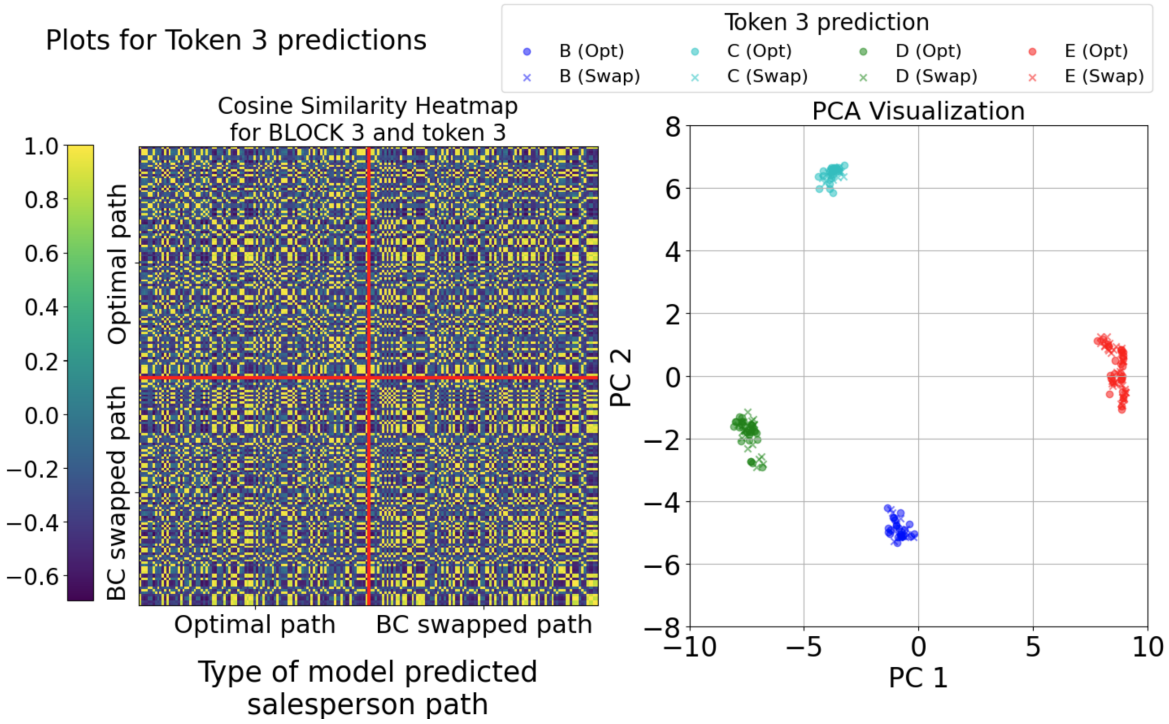


Fig. 3.33 Letter 3, Block 3

Note that the letters cluster in Figure 3.33. This is because for the very last token in the last block, there is one city remaining to traverse, thus it is clear that this city must be the output.

This is where the project had to stop due to time constraints. Consequently, we were unable to demonstrate mechanistically that goals are indeed present in the network's activations. Nevertheless, the observed clusterings give hope for future investigations, which could indicate the presence of goals within the network's representations.

Chapter 4

Conclusion

This work investigated if an imitation learning agent can inherit the goals of an expert which solves the traveling salesperson problem. The notion of goal-directedness was defined and examined, focusing on its presence in the behaviour of LMs, namely microGPT and GPT-2.

To explore this problem, behavioural experiments were conducted across various TSP setups. This was achieved by making the evaluation set be out-of-distribution in distinct ways while preserving the correctness of the goals present in the training set. This approach allowed us to investigate generalisation in different TSP setups. First, a baseline model performance was established. Then different segments of the grid were omitted or shown in a few number of examples during training, but the model was evaluated only on examples containing these segments. The main findings from these experiments suggest that the model is acting in a goal-directed manner, as it successfully solves TSP with cities placed on previously unseen grid segments.

The goal-directedness of the model was also examined in presence of two different types of noise. When random noise was added, the model demonstrated the ability to follow the goal even with substantial amounts of noise. When systematic noise was added, the model showed tendency to follow whichever goal was most present in the training dataset. By doing this, the model was able not only to match but also to exceed the training agent performance. The model generalised overconfidently on the evaluation set and an investigation of why this happens was conducted by looking at the per-token loss of each prediction the model makes.

Finally, a mechanistic interpretability analysis of model activations in microGPT with various inputs was conducted to try to uncover any emergent goal representations within the model weights. While the findings from this analysis are encouraging, there is still potential for future work.

References

Alammar, J. (2019). The illustrated gpt-2 (visualizing transformer language models). Accessed: 2024-05-21.

Albanie, S. (2023). Lecture notes in deep learning for computer vision - transformers.

Andreas, J. (2022). Language models as agent models.

Applegate, D., Bixby, R., Chvatal, V., and Cook, W. (2006). Concorde tsp solver.

Arjovsky, M., Bottou, L., Gulrajani, I., and Lopez-Paz, D. (2020). Invariant risk minimization.

Arpit, D., Jastrzębski, S., Ballas, N., Krueger, D., Bengio, E., Kanwal, M. S., Maharaj, T., Fischer, A., Courville, A., Bengio, Y., and Lacoste-Julien, S. (2017). A closer look at memorization in deep networks.

Dennett, D. (2009). 339 Intentional Systems Theory. In *The Oxford Handbook of Philosophy of Mind*. Oxford University Press.

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., DasSarma, N., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. (2021). A mathematical framework for transformer circuits. *Transformer Circuits Thread*. https://transformer-circuits.pub/2021/framework/index.html.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Hendrycks, D. and Gimpel, K. (2023). Gaussian error linear units (gelus).

HF Canonical Model Maintainers (2022). gpt2 (revision 909a290).

Karpathy, A. (2022). minGPT: Minimalist Generative Pre-trained Transformer.

Krueger, D., Caballero, E., Jacobsen, J.-H., Zhang, A., Binas, J., Zhang, D., Priol, R. L., and Courville, A. (2021). Out-of-distribution generalization via risk extrapolation (rex). In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5815–5826. PMLR.

Li, K., Hopkins, A. K., Bau, D., Viégas, F., Pfister, H., and Wattenberg, M. (2023). Emergent world representations: Exploring a sequence model trained on a synthetic task.

Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization.

Lovering, C., Forde, J. Z., Konidaris, G., Pavlick, E., and Littman, M. L. (2022). Evaluation beyond task performance: Analyzing concepts in alphazero in hex.

Lubana, E. S., Bigelow, E. J., Dick, R. P., Krueger, D., and Tanaka, H. (2023). Mechanistic mode connectivity.

Nanda, N., Lee, A., and Wattenberg, M. (2023). Emergent linear representations in world models of self-supervised sequence models.

Orseau, L., McGill, S. M., and Legg, S. (2018). Agents and devices: A relative definition of agency.

Quintas, L. V. and Supnick, F. (1965). On some properties of shortest hamiltonian circuits. *The American Mathematical Monthly*, 72(9):977–980.

Radford, A. and Narasimhan, K. (2018). Improving language understanding by generative pre-training.

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

Shimi, A., Campolo, M., and Collman, J. (2021). Literature review on goal-directedness. Accessed: 2024-05-22.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2023). Attention is all you need.

Xing, Z. and Tu, S. (2020). A graph neural network assisted monte carlo tree search approach to traveling salesman problem. *IEEE Access*, 8:108418–108428.

Appendix A

GPT-2

GPT-2 (Radford et al. (2019)) is an auto-regressive model built using transformer (Vaswani et al. (2023)) decoder-only blocks. It outputs one token at a time. The outputted token is added to the sequence of inputs.

Decoder-Only Block

The GPT-2 decoder-only blocks, as shown in Figure A.1 are based on the OpenAI GPT model Radford and Narasimhan (2018) with slight modifications. Please see Radford et al. (2019) for exact specifications.

First, a tokeniser converts raw text to a sequence of N integer tokens according to a vocabulary of $\mathcal{V}$ possible elements. Every token has a token embedding and positional embedding, each of size D. The token embedding is a list of numbers representing that token and captures some of its meaning. The positional embedding indicates the order of the input tokens to the transformer blocks. The token and positional embeddings are added for each token and stacked to form a matrix $X \in N \times D$. X is passed to the first block A.1.

Self-attention is what allows the different token embedding to communicate with each other. There are three significant components. Each position embedding gets projected to a key $K = XW^K$, a query $Q = XW^Q$ and a value $V = XW^V$ via matrices $W^Q \in \mathbb{R}^{D \times d_k}$, $W^K \in \mathbb{R}^{D \times d_k}$ and $W^V \in \mathbb{R}^{D \times d_v}$ respectively, where d_k is the dimension of queries and keys, while d_v is the dimension of values Albanie (2023).

Scaled dot product attention is the computed as $Y = \texttt{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \in \mathbb{R}^{N \times d_v}$.

Multi-Head Attention means splitting the previous operation over H multiple operations (heads), which can be executed in parallel. Hence, for each $h \in \{1, \ldots, H\}$, compute $K_h = XW_h^K$, $Q_h = XW_h^Q$ and $V_h = XW_h^V$, where now $d_k = d_v = \frac{D}{H}$.

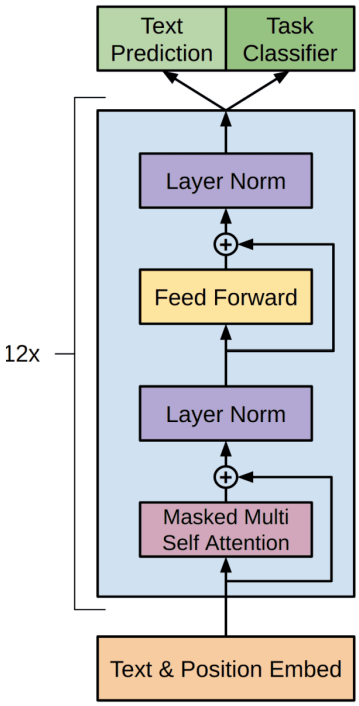


Fig. A.1 Transformer architecture. Figure by Radford and Narasimhan (2018).

Then calculate $\texttt{head}_h = \texttt{softmax}\left(\frac{Q_h K_h^T}{\sqrt{d_k}}\right) V_h$. The heads are concatenated and projected using W^O, resulting in $\texttt{MultiHead}(X) = \texttt{Concat}(\texttt{head}_1, \ldots, \texttt{head}_h) W^O$.

The Feed Forward layer is implemented via a 2-layer Multi-Layer Perceptron (MLP) applied to each embedding $x \in \mathbb{R}^D$ individually $\texttt{MLP}(x) = W_2 \sigma(W_1 x + b_1) + b_2$, where σ is the GeLU non-linearity Hendrycks and Gimpel (2023). An expansion factor of four is used, hence $W_1 \in \mathbb{R}^{D \times 4D}$ and $W_2 \in \mathbb{R}^{4D \times D}$.

The layer norm layers operate as $y = \frac{x - \mathbb{E}[x]}{\sqrt{Var[x] + \epsilon}} \cdot \gamma + \beta$, where γ and β are learnable parameters. $\mathbb{E}[x]$ and $Var[x]$ are computed along the embedding dimension D.

Note also the two residual connections He et al. (2015) which help gradients flow, avoid the vanishing gradient problem and help with preconditioning.

The Masked part of the Multi-Head Attention indicates that only causal attention is allowed, restricting the current token to attend solely to preceding tokens. This is done by multiplying the scores (before softmax) with a $N \times N$ matrix whose below diagonal entries are one above diagonal entries are $-\infty$.

Cross Entropy

A language model aims to learn a distribution Q which should be *close* to the true distribution P. Cross Entropy defines how close P and Q are. Cross entropy is defined as:

$$H(P,Q) = \mathbb{E}_P[-logQ] \tag{A.1}$$

For discrete P and Q this can be written as

$$H(P,Q) = -\sum_x P(x)\, logQ(x) \tag{A.2}$$

This can be rewritten as:

$$\begin{aligned} H(P,Q) &= -\sum_x P(x)\left[\log P(x) + \log\frac{Q(x)}{P(x)}\right] \\ &= -\sum_x P(x)\log P(x) - \sum_x P(x)\log\frac{Q(x)}{P(x)} \\ &= H(P) + D_{KL}(P||Q) \end{aligned} \tag{A.3}$$

where H is the Shannon entropy. It measures the information content of a distribution or how much we learn from sampling.

The Kullback-Leibler divergence D_{KL} measures how much one distribution diverges from the other. Another interpretation is that it measures how likely samples drawn from Q would be mistaken for samples drawn from P.

During training, the Cross Entropy will penalise distributions that make wrong class predictions, with a higher penalty for more inaccurate predictions. The objective of training the model is to minimise the Cross Entropy.

Appendix B

TSP Setups

B.1 Concorde TSP Solver

During most of the project, the TSP datasets were generated using the Concorde TSP solver Applegate et al. (2006). Issues with the Concorde generated data that hinder model learning were discovered late in the project.

The Concorde solver uses a distance function where all point to point distances are integer-valued. This means the Euclidean distance is rounded to the nearest integer. As Concorde solves the TSP in an integer Euclidean metric, the solver may fail to find the optimal solution. Concorde would be suitable for usage if, for example, the cities were placed on a 1000000×1000000 grid.

The generated salesperson's route using Concorde is sometimes clockwise and sometimes anticlockwise, with no discernible preference between the two. This hampers model learning as noted in Section 2.2.

During the project, significant effort was dedicated towards finding a suitable learning rate, weight decay, TSP problem setup and various dataset sizes generated with and without Concorde Applegate et al. (2006), as well as a suitable GPT-2 model size. The report presents the main findings, without detailing the intermediary design decisions, which were based on intuitions rather than scientifically-based reasons for why certain setups were successful while others were not.

To name a few, we experimented with models ranging in size from nanoGPT (with $|\mathcal{V}| = 13$ and approximately $134k$ parameters) up to a 1.5B parameter GPT-2 with $|\mathcal{V}| = 50257$. We also tried TSP problems with varying configurations, such as 6, 9, or 10 cities on a 9×9 grid, 7 cities on a 4×4 grid, and 5 cities on a 6×5 grid, among others.

Appendix C

Activations in Block two of microGPT

We also noticed for one block's representations, that each point changes all four colours, as shown in Figures C.1, C.2, C.3 and C.4 and it will always stay either a 'x' or 'o'. This shows that for each input, the salesperson visits each of the cities `B`, `C`, `D` and `E` once.

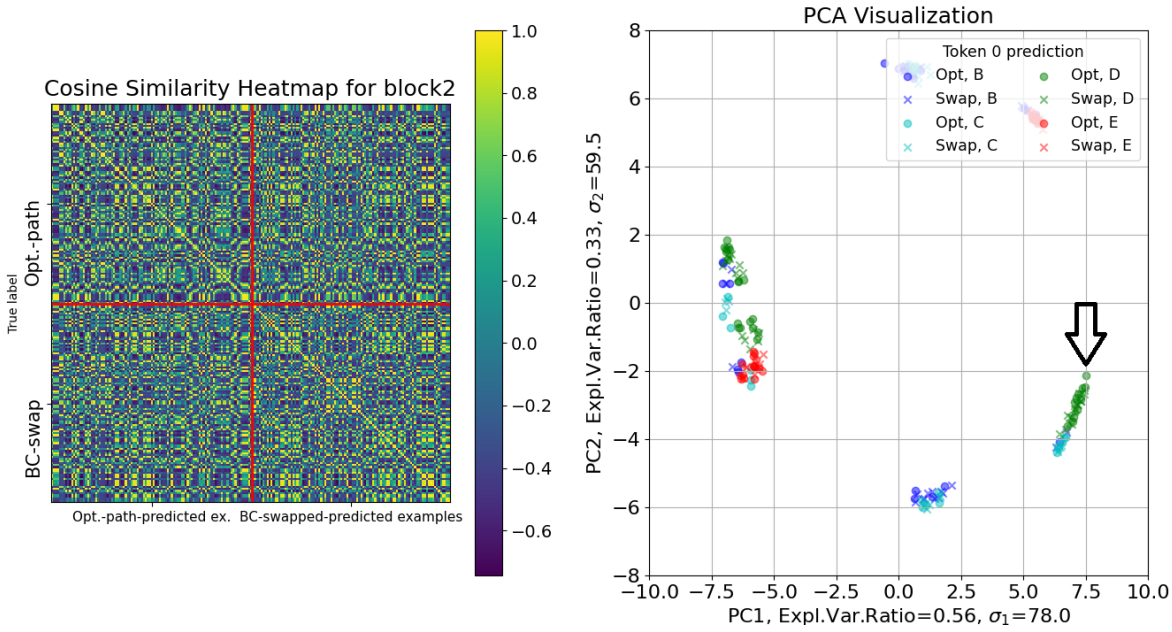


Fig. C.1 Letter 0, Block 2

Overall we hypothesise the model is somehow sorting the activations after each block (Figures 3.30, 3.31, 3.32, 3.33) and after each processed token (Figures C.1, C.2, C.3 and C.4).

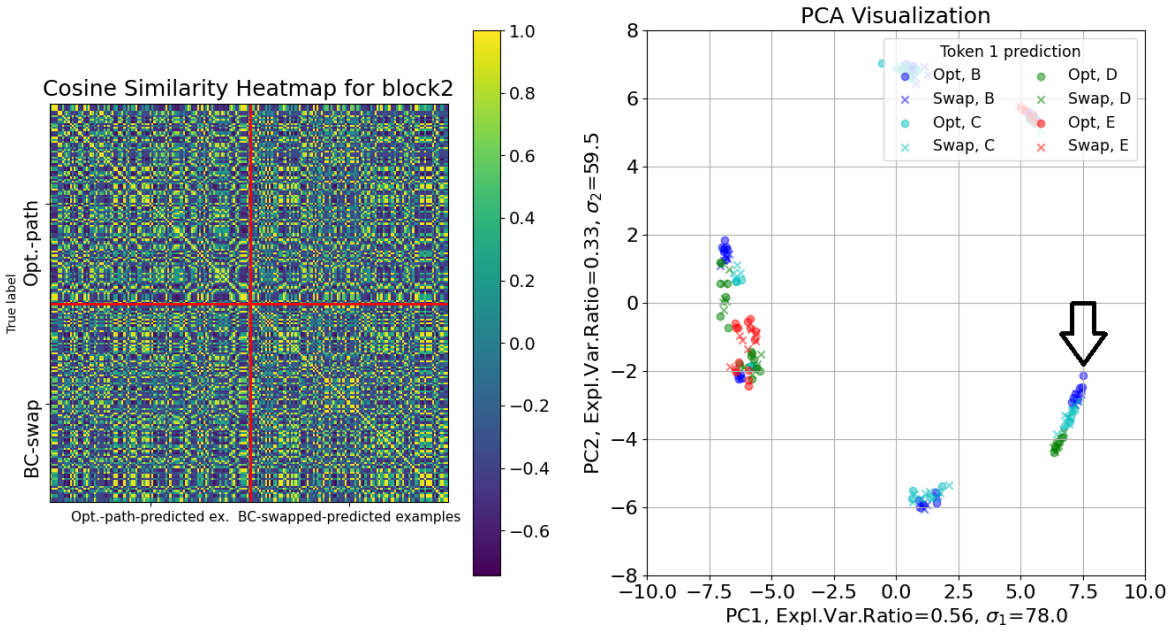


Fig. C.2 Letter 1, Block 2

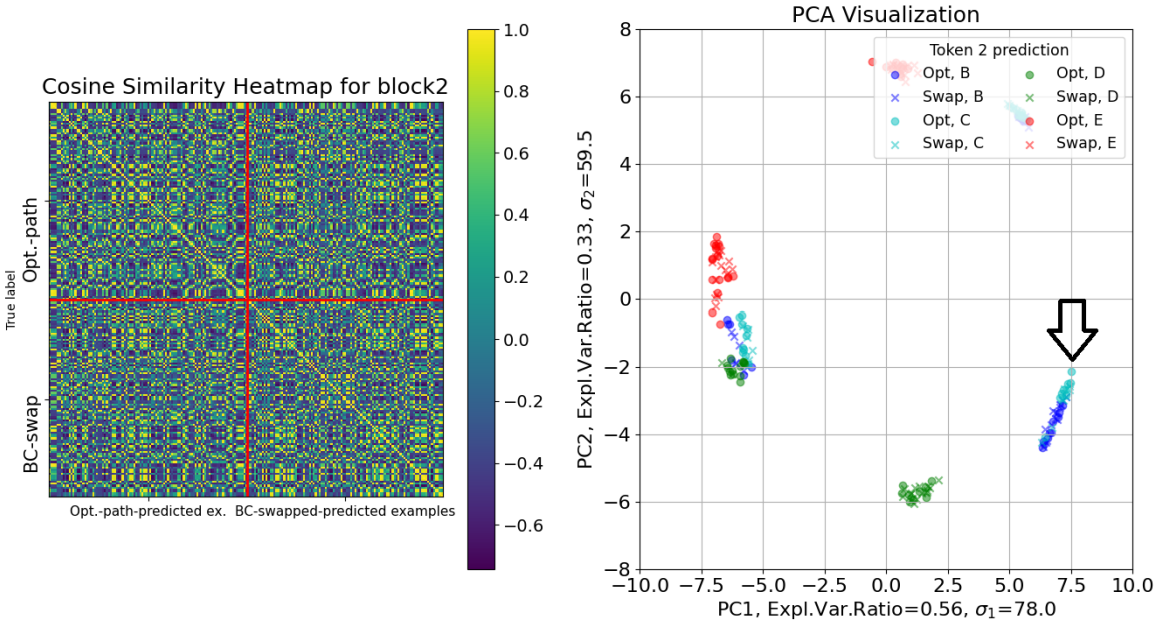


Fig. C.3 Letter 2, Block 2

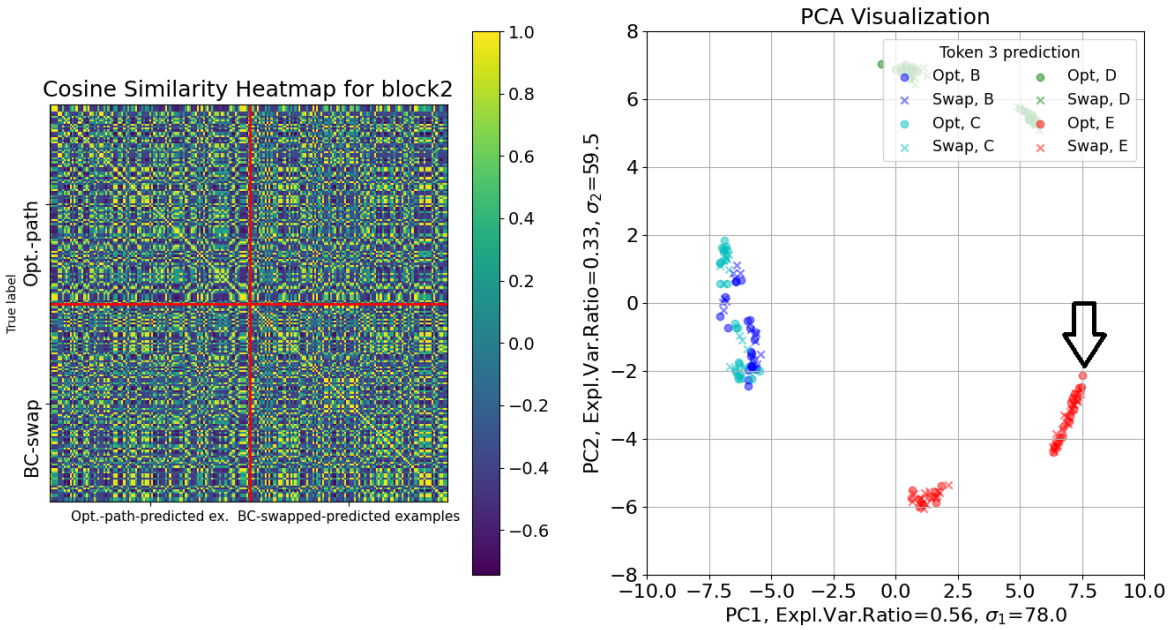


Fig. C.4 Letter 3, Block 2

Appendix D

Evaluation of Risk Assessment

In my Risk Assessment from Michaelmass 2023, I wrote "Project may produce an AI agent which could pursue dangerous goals". Fortunately, this scenario did not materialise, largely due to the small scale of my AI language model and the specific problems it was designed to solve. However, my project made a contribution towards defining what is meant for a language model to be goal-directed. This contribution takes place in the broader context of AI safety and alignment research, as it represents a step towards ensuring that AI models are safe and alignable with human values, which remains an active and vital area of research.

On a personal level, the Safety Department's guideline proved useful and I was following it at first. Upon reflection, I realised I neglected to prioritise proper computer usage to prevent physical and mental strain. While initially following safety guidelines, I slipped back into poor habits, leading to discomfort and stress. Moving forward, I must be mindful of maintaining good posture, adjusting screen height, and taking regular breaks to ensure long-term well-being during prolonged computer use in my career.