SCIENCE CHINA
Information Sciences

• Supplementary File •

The Impact of Task Similarity on Performance Drift of LLMs: A Theoretical and Empirical Analysis

Xuanming NI[1], Qiaochu ZHAO[1], Song HUANG[1*] & Lian YU[1]

[1]*School of Software and Microelectronics, Peking University, Beijing* 102600, *China*

Appendix A Theoretical results and proofs

First of all, some notations are defined in order to simplify the expressions. Hereafter we denote by $[n]$ the set $\{1,\ldots,n\}$, denote by $\mathbf{1}(E)$ the indicator function of event E, denote by $len(s)$ the number of bits of string s, denote by $s(1:l)$ the first l elements of string s, and denote by $\mathcal{U}(A)$ the uniform distribution on set A.

Our theoretical work is based on the notion that the natural language distribution can be viewed as consisting of numerous subpopulations [3]. Suppose there are N subpopulations in total, each of which is independent to the others, then the entire distribution can be represented by a mixture of distributions of subpopulations, as follows.

$$M(x)=\sum_{i\in[N]}\alpha_i M_i(x),\quad \sum_{i\in[N]}\alpha_i=1. \tag{A1}$$

Specifically, for the causal language models, whose objective is to predict the distribution of the next symbol given a prefix of symbols, we adopt an assumption similar to that used in some previous work [1, 2]. Namely, the next symbol prediction objective is simplified to predicting the next bit 0 or 1 given a prefix consisting of only 0s and 1s. Furthermore, we assume that the distribution of each subpopulation is as follows.

Definition 1 (Binary symmetric channel [4])**.** The result of flipping each bit of a string s independently with probability δ is denoted by $BSC_\delta(s)$, with each bit of it defined as follows.

$$BSC_\delta(s)(k)=\begin{cases}s(k), & r_k>\delta\\ 1-s(k), & r_k\leqslant\delta\end{cases} \tag{A2}$$

where $k\in[len(s)]$ and $r_1,\ldots,r_l\overset{\text{iid}}{\sim}\mathcal{U}([0,1])$. For the sake of brevity, we will use the notation $z\sim BSC_\delta(s)$ to indicate that z follows the same distribution as $BSC_\delta(s)$.

Assumption 1 ([1, 2])**.** The ith subpopulation is featured by a reference string $c_i\sim\mathcal{U}(\{0,1\}^d)$. A sample z from the ith subpopulation is drawn by first selecting $l\sim\mathcal{U}([n])$, and then flipping each bit of $c_i(1:l)$ independently and randomly with the same probability $\delta<\frac{1}{2}$.

$$z=BSC_\delta(c_i(1:l)),\quad l\sim\mathcal{U}([n]). \tag{A3}$$

The only difference between our problem setting and that presented in [1, 2] is that each sample z is treated as a string, rather than a pair of prefix and label. The algorithm A will perform the next symbol prediction task on every pair of prefix $z(1:s-1)$ and label $z(s)$ for all $s\leqslant len(z)$. This setting is a natural generalization of the original problem setting in [1, 2] and is more suitable for simulating the pre-training process of causal language models.

Let us consider a dataset Z consisting of n samples drawn independently and identically from the entire population, $Z=\{z_1,\ldots,z_n\}$. An algorithm A takes dataset Z as input and outputs a predictor $h=A(Z)$. The predictor h can then be applied to any prefix z to predict the next bit $y=h(z)$. To evaluate the accuracy of an algorithm, the error rate of algorithm A on the jth subpopulation is defined as follows.

Definition 2 (Error rate on subpopulation j given dataset Z)**.** Given dataset Z, the averaged error rate of algorithm A on the jth subpopulation is defined as

$$\overline{\text{err}}_j(c_j,A\mid Z)=\mathop{\mathbb{E}}_{h\sim A(Z)}\mathop{\mathbb{E}}_{z\sim M_j}\mathbf{1}(h(z)\neq c_j(len(z)+1)). \tag{A4}$$

In more general cases, the dataset Z is unknown. The following definition generalizes Definition 2 to the situation where only the occurrences and similarities are known.

* Corresponding author (email: huangsong@ss.pku.edu.cn)

Definition 3 (Averaged error rate on subpopulation j)**.** Denote $\alpha = (\alpha_1, \ldots, \alpha_N)$ the probabilities of occurrence of all the tasks, and $C = (r(i,j))_{1 \leqslant i,j \leqslant N}$ the matrix of similarity between tasks. Denote by $D(\alpha, C)$ the distribution of one sample from dataset given α and C. Then the distribution of Z is $D^n(\alpha, C)$. The expected error rate of any algorithm A on the jth subpopulation with respect to the dataset distribution is defined as

$$\overline{\mathrm{err}}_j(\alpha, C, A) = \mathop{\mathbb{E}}_{Z \sim D^n(\alpha, C)} \overline{\mathrm{err}}_j(c_j, A \mid Z). \tag{A5}$$

The purpose of the following discussion is to address the question of the impact of a single sample if the algorithm erroneously classifies it as belonging to a different subpopulation. The theoretical results presented here indicate that this will have a negative impact on the lower bound of the algorithm's error rate on the subpopulation to which the sample appears to belong.

In theoretical analysis, the algorithm in question must satisfy certain constraints to prevent it from having unlimited capacity. First, it is assumed that the algorithm would mistake a sample for belonging to another subpopulation only when this sample is sufficiently similar to that subpopulation. Therefore, it is necessary to define similarity in our settings.

Definition 4. The similarity of two subpopulations i and j is defined by comparing the corresponding reference strings c_i and c_j.

$$r(i,j) = \frac{1}{d} \sum_{k=1}^{d} \mathbf{1}(c_i(k) = c_j(k)). \tag{A6}$$

Similarly, for a sample z_0 drawn from M_i and another subpopulation j, similarity can be defined as

$$r_j(z_0) = \frac{1}{len(z_0)} \sum_{k=1}^{len(z_0)} \mathbf{1}(z_0(k) = c_j(k)). \tag{A7}$$

It is not hard to see that $\mathbb{E}_{z_0 \sim M_i} r_j(z_0) = (1 - 2\delta) r(i,j) + \delta$.

In order to answer the question of to what extent the algorithm A will incorrectly identify a sample z_0 as belonging to a different subpopulation j, we must make the following assumption about the behavior of the algorithm A.

Assumption 2. An algorithm A will incorrectly identify a sample z_0 from subpopulation i as belonging to subpopulation j with a probability proportional to the similarity $r_j(z_0)$.

$$p_{\mathrm{mis}} = k_{\mathrm{mis}} r_j(z_0). \tag{A8}$$

In addition, it is necessary for the performance of algorithm A to be affected after an erroneous identification of the subpopulation of a sample. The following assumption is an intuitive quantification of the extent to which the algorithm is affected.

Assumption 3. Given a specific dataset $Z = \{z_1, \ldots, z_l\} \cup \{z_0\}$, where $z_1, \ldots, z_l \sim M_j$ and $z_0 \sim M_i, i \neq j$, for every prefix z that $len(z) \geqslant len(z_0) - 1$, it is assumed that z_0 have no impact on the distribution of $h(z)$. For any prefix z such that $len(z) = k < len(z_0) - 1$, denoting $Z^* = Z \setminus \{z_0\}$, A^* the optimal algorithm for Z^*, the following inequalities hold.

$$\mathop{\mathbb{E}}_{\substack{h \sim A(Z) \\ z \sim BSC_\delta(c_j(1:k))}} \mathbf{1}(h(z) \neq c_j(k+1) \mid z_0(k+1) \neq c_j(k+1)) \geqslant \mathop{\mathbb{E}}_{\substack{h \sim A^*(Z^*) \\ z \sim BSC_\delta(c_j(1:k))}} \mathbf{1}(h(z) \neq c_j(k+1)) + \lambda_l, \tag{A9}$$

$$\mathop{\mathbb{E}}_{\substack{h \sim A(Z) \\ z \sim BSC_\delta(c_j(1:k))}} \mathbf{1}(h(z) = c_j(k+1) \mid z_0(k+1) = c_j(k+1)) \leqslant \mathop{\mathbb{E}}_{\substack{h \sim A^*(Z^*) \\ z \sim BSC_\delta(c_j(1:k))}} \mathbf{1}(h(z) = c_j(k+1)) + \mu_l, \tag{A10}$$

where λ_l and μ_l depends only on the number l. Furthermore, it is assumed that $\lambda_l \gg \mu_l$ when $l \geqslant n_0$ and $n_0 = o(d)$.

It is then possible to deduce the lower bound of the excess risk introduced by an out-of-subpopulation sample.

Theorem A1. Given a specific dataset $Z = \{z_1, \ldots, z_l\} \cup \{z_0\}$, where $z_1, \ldots, z_l \sim M_j$ and $z_0 \sim M_i, i \neq j$, the excess risk of any algorithm A compared to $Z^* = Z \setminus \{z_0\}$ and the optimal algorithm A^* for Z^* can be bounded as follows.

$$\overline{\mathrm{err}}_j(c_j, A \mid Z) \geqslant \mathop{\mathbb{E}}_{\substack{h \sim A^*(Z^*) \\ z \sim M_j}} \mathbf{1}(h(z) \neq c_j(len(z)+1)) + \frac{k_{\mathrm{mis}} len(z_0)(\lambda_l - (\lambda_l + \mu_l) r_j(z_0)) r_j(z_0)}{d}. \tag{A11}$$

Proof. First write down $\overline{\mathrm{err}}_j(c_j, A \mid Z)$ in terms of probability.

$$\overline{\mathrm{err}}_j(c_j, A \mid Z) = \mathop{\mathbb{E}}_{h \sim A(Z)} \mathop{\mathbb{E}}_{z \sim M_j} \mathbf{1}(h(z) \neq c_j(len(z)+1)) = \mathop{\mathbb{E}}_{h \sim A(Z)} \frac{1}{d} \sum_{k=0}^{d-1} \mathop{\mathbb{E}}_{z \sim BSC_\delta(c_j(1:k))} \mathbf{1}(h(z) \neq c_j(k+1)). \tag{A12}$$

Note that (A10) in Assumption 3 is equivalent to

$$\mathop{\mathbb{E}}_{\substack{h \sim A(Z) \\ z \sim BSC_\delta(c_j(1:k))}} \mathbf{1}(h(z) \neq c_j(k+1) \mid z_0(k+1) = c_j(k+1)) \geqslant \mathop{\mathbb{E}}_{\substack{h \sim A^*(Z^*) \\ z \sim BSC_\delta(c_j(1:k))}} \mathbf{1}(h(z) \neq c_j(k+1)) - \mu_l. \tag{A13}$$

Then for any $0 \leqslant k \leqslant len(z_0) - 1$ such that $z_0(k+1) \neq c_j(k+1)$, we have

$$\mathop{\mathbb{E}}_{\substack{h\sim A(Z)\\ z\sim BSC_\delta(c_j(1:k))}} \mathbf{1}(h(z) \neq c_j(k+1)) \geqslant k_{\mathrm{mis}} r_j(z_0)\Big(\mathop{\mathbb{E}}_{\substack{h\sim A^*(Z^*)\\ z\sim BSC_\delta(c_j(1:k)}} \mathbf{1}(h(z) \neq c_j(k+1)) + \lambda_l \Big) \tag{A14}$$

$$+ (1 - k_{\mathrm{mis}} r_j(z_0)) \mathop{\mathbb{E}}_{\substack{h\sim A^*(Z^*)\\ z\sim BSC_\delta(c_j(1:k)}} \mathbf{1}(h(z) \neq c_j(k+1)) \tag{A15}$$

$$= \mathop{\mathbb{E}}_{\substack{h\sim A^*(Z^*)\\ z\sim BSC_\delta(c_j(1:k)}} \mathbf{1}(h(z) \neq c_j(k+1)) + k_{\mathrm{mis}} r_j(z_0)\lambda_l. \tag{A16}$$

Similar inequality can be derived for $0 \leqslant k \leqslant len(z_0) - 1$ such that $z_0(k+1) = c_j(k+1)$.

$$\mathop{\mathbb{E}}_{\substack{h\sim A(Z)\\ z\sim BSC_\delta(c_j(1:k))}} \mathbf{1}(h(z) \neq c_j(k+1)) \geqslant \mathop{\mathbb{E}}_{\substack{h\sim A^*(Z^*)\\ z\sim BSC_\delta(c_j(1:k)}} \mathbf{1}(h(z) \neq c_j(k+1)) - k_{\mathrm{mis}} r_j(z_0)\mu_l. \tag{A17}$$

Combine (A16) and (A17), we have the inequality stated in this theorem. The proof is complete.

Theorem A1 provides a lower bound. Obviously, $\frac{\lambda_l}{\lambda_l+\mu_l}$ is close to 1 when l is large.

Theorem A1 demonstrates the impact of a single sample drawn from a different subpopulation. In a more general context, the dataset Z is generated by the distribution $D^n(\alpha, C)$. Consequently, it is overly simplistic to assume that a subpopulation can be represented by a single sample.

The following assumption posits that if the algorithm A is provided with multiple inputs from subpopulation i, there exists a reduction to the situation of Theorem A1.

Assumption 4. Given a specific dataset $Z = \{z_1, \ldots, z_l\} \cup \{z'_1, \ldots, z'_m\}$, where $z_1, \ldots, z_l \sim M_j$ and $z'_1, \ldots, z'_m \sim M_i, i \neq j$, denote $Z^* = \{z_1, \ldots, z_l\}$ and A^* the optimal algorithm. Without loss of generality, suppose z'_m is the longest string in $\{z'_1, \ldots, z'_m\}$. Then

$$\mathop{\mathbb{E}}_{\substack{h\sim A(Z)\\ z\sim M_j}} \mathbf{1}(h(z) \neq c_j(len(z)+1)) \geqslant \mathop{\mathbb{E}}_{\substack{h\sim A^*(Z^*\cup\{z'_m\})\\ z\sim M_j}} \mathbf{1}(h(z) \neq c_j(len(z)+1)). \tag{A18}$$

With Assumption 4, it is now possible to deduce the lower bound of the excess risk of the algorithm in a more general case.

Theorem A2. Suppose the dataset Z follows the distribution $D^n(\alpha, C)$. Denote by A^* the optimal algorithm and denote by $OPT(A^*)$ the error rate of A^* with input dataset $\{z \in Z \mid z \text{ is from subpopulation } j\}$. Then the averaged error rate of any algorithm A that satisfies all the assumptions above can be lower bounded as

$$\overline{\mathrm{err}}_j(\alpha, C, A) \geqslant OPT(A^*) + f_{n,d,\alpha,C}(r(i,j)), \tag{A19}$$

where $f_{n,d,\alpha,C}$ has positive maximum in $[0,1]$ and takes its maximum near but lower than $\frac{1}{2}$.

Proof. First write down $\overline{\mathrm{err}}_j(\alpha, C, A)$ in terms of probability.

$$\overline{\mathrm{err}}_j(\alpha, C, A) = \sum_{\substack{1\leqslant l\leqslant n-1\\ 1\leqslant m\leqslant n-1\\ l+m\leqslant n}} \binom{n}{l}\binom{n-l}{m} \alpha_j^l \alpha_i^m (1-\alpha_j-\alpha_i)^{n-l-m} \sum_{l_0=1}^{d} p_m(l_0) \overline{\mathrm{err}}_j(c_j, A, Z \mid l, m, l_0) \tag{A20}$$

$$+ \Pr[l = 0 \vee m = 0] OPT(A^*), \tag{A21}$$

where l is the number of samples from subpopulation j, m is the number of samples from subpopulation i, l_0 is the length of the longest string from subpopulation i, $p_m(l_0)$ is the probability that the longest string from subpopulation i has a length of l_0, and $\overline{\mathrm{err}}_j(c_j, A, Z \mid l, m, l_0)$ is the averaged error rate of algorithm A conditioned on l, m, l_0 fixed. It is not hard to see that

$$p_m(l_0) = (\frac{l_0}{d})^m - (\frac{l_0-1}{d})^m. \tag{A22}$$

By Theorem A1 and Assumption 4, we have

$$\overline{\mathrm{err}}_j(c_j, A, Z \mid l, m, l_0) \geqslant OPT(A^*) + \frac{k_m l_0}{d}(\lambda_l \,\mathbb{E}[r_j \mid r(i,j), l_0] - (\lambda_l + \mu_l)\,\mathbb{E}[r_j^2 \mid r(i,j), l_0]). \tag{A23}$$

Here in (A23), $\mathbb{E}[r_j \mid r(i,j), l_0]$ means the expectation of $r_j(z')$ given $len(z') = l_0$ and $r(i,j)$. The same is for $\mathbb{E}[r_j^2 \mid r(i,j), l_0]$. It is clear that $\mathbb{E}[r_j \mid r(i,j), l_0] = (1-2\delta)r(i,j) + \delta$. To calculate $\mathbb{E}[r_j^2 \mid r(i,j), l_0]$, we define the following notations.

$$\alpha = d \cdot r(i,j), \quad \beta = \sum_{k=1}^{l_0} \mathbf{1}(c_i(k) = c_j(k)). \tag{A24}$$

Then β is a random variable depedent only on $r(i,j)$ and l_0. Actually, it is not hard to see that β follows a hypergeometric distribution with mean $r(i,j)$ and variance $\frac{\alpha l_0(d-\alpha)(d-l_0)}{d^2(d-1)}$. The relation between β and $\mathbb{E}[r_j^2 \mid r(i,j), l_0]$ is

$$\mathbb{E}[r_j^2 \mid r(i,j), l_0] = \sum_{\beta'=\max(0,\alpha+l_0-d)}^{\min(l_0,\alpha)} \Pr[\beta=\beta'](\frac{(1-\delta)\beta' + (l_0-\beta')\delta}{l_0})^2, \tag{A25}$$

$$= \frac{(1-2\delta)^2}{l_0^2}\,\mathbb{E}\,\beta^2 + \frac{2\delta(1-\delta)}{l_0}\,\mathbb{E}\,\beta + \delta^2. \tag{A26}$$

Substitute the mean and variance of β, we have

$$\mathbb{E}[r_j^2 \mid r(i,j), l_0] = \frac{(1-2\delta)^2(d-l_0)}{l_0(d-1)} r(i,j)(1-r(i,j)) + (1-2\delta)^2 r^2(i,j) + 2\delta(1-2\delta)r(i,j) + \delta^2. \tag{A27}$$

Now to simplify (A20), denote

$$h_{l,m} = \binom{n}{l}\binom{n-l}{m}\alpha_j^l \alpha_i^m (1-\alpha_j-\alpha_i)^{n-l-m}, \quad g_m = \sum_{l_0=1}^{d} p_m(l_0)\frac{l_0 k_m}{d}\frac{(1-2\delta)^2(d-l_0)}{l_0(d-1)}. \tag{A28}$$

Combine (A20), (A23) and (A27), we have

$$\overline{\mathrm{err}}_j(\alpha, C, A) \geqslant OPT(A^*) - \sum_{l=1}^{n-1}\sum_{m=1}^{n-l} h_{l,m} g_m (\lambda_l + \mu_l) r(i,j)(1-r(i,j)) \tag{A29}$$

$$- \sum_{l=1}^{n-1}\sum_{m=1}^{n-l} h_{l,m} \sum_{l_0=1}^{d} p_m(l_0) l_0 \frac{k_m}{d}\Big((\lambda_l+\mu_l)\delta^2 - \lambda_l\delta \tag{A30}$$

$$+ (\lambda_l+\mu_l)(1-2\delta)^2 r^2(i,j) - \lambda_l(1-2\delta)r(i,j) + 2(\lambda_l+\mu_l)\delta(1-2\delta)r(i,j)\Big). \tag{A31}$$

Note that the part (A29) always takes its maximum value at $r(i,j)=\frac{1}{2}$, and the part (A30) is a constant. For part (A31), it will takes its maximum value at

$$r(i,j) = \frac{\lambda}{2(1-2\delta)(\lambda+\mu)} - \frac{\delta}{1-2\delta}, \tag{A32}$$

where

$$\lambda = \sum_{l=1}^{n-1}\sum_{m=1}^{n-l} h_{l,m} \sum_{l_0=1}^{d} p_m(l_0) l_0 \frac{k_m}{d}\lambda_l, \quad \mu = \sum_{l=1}^{n-1}\sum_{m=1}^{n-l} h_{l,m} \sum_{l_0=1}^{d} p_m(l_0) l_0 \frac{k_m}{d}\mu_l. \tag{A33}$$

By the fact that $\lambda < \lambda+\mu$, (A32) is always lower than $\frac{1}{2}$. And by Assumption 3, λ is close to $\lambda+\mu$, then (A32) is close to $\frac{1}{2}$. Now the proof is complete.

Theorem A2 states that any arbitrary algorithm that satisfies the assumptions would have difficulty in achieving the optimal error rate by a factor related to task similarity, regardless of the type of the algorithm and the training process. However, this theorem does not imply that an algorithm cannot avoid performance degradation when trained on a mixture of data. Looking back at Assumption 2 and Assumption 3, the coefficients in these assumptions are what constrains the algorithm's capabilities. For instance, an algorithm that has no difficulty in distinguishing the subpopulation to which each sample belongs will not be subject to such limitations. This understanding also provides insights into potential avenues for mitigating such degradation of the model. These include strategies to enhance the model's ability to distinguish different subpopulations and to expand its capacity to memorize multiple subpopulations without mutual influence.

Appendix B Details of experiments on synthetic data

The following set of experiments is conducted to validate Theorem A2 on synthetic data consisting of binary strings. A transformer model is trained on a training set consisting of data from a base distribution and from another different distribution. Thereafter, this model is evaluated on a test set from the base distribution. According to Theorem A2, when the similarity between the two distributions is close to and less than $\frac{1}{2}$, the model's loss on the test set should be the worst.

Generating data from distributions (subpopulations). Each distribution is featured by a reference string of length L. To draw a sample from a distribution with reference string c, we first select an integer $l \in [1, L]$ at random and truncate c to the length l. Then, each bit of the truncated string is flipped with probability $\delta < 1/2$ independently. The resulting string is a sample from the distribution. When multiple samples are drawn from the same distribution, each sample is drawn independently.

How is every distributions generated. To determine a distribution is equivalent to determining a reference string c. First, the reference string of the base distribution c_0 is constructed by randomly assigning each bit of c_0 to either the value 0 or 1. Subsequently, the distribution of similarity r to the base distribution is constructed through the following process. Randomly selecting $\lfloor rL \rfloor$ bits of c_0 without overlap and flipping them, the resulting string of length L is designated as the reference string of this distribution.

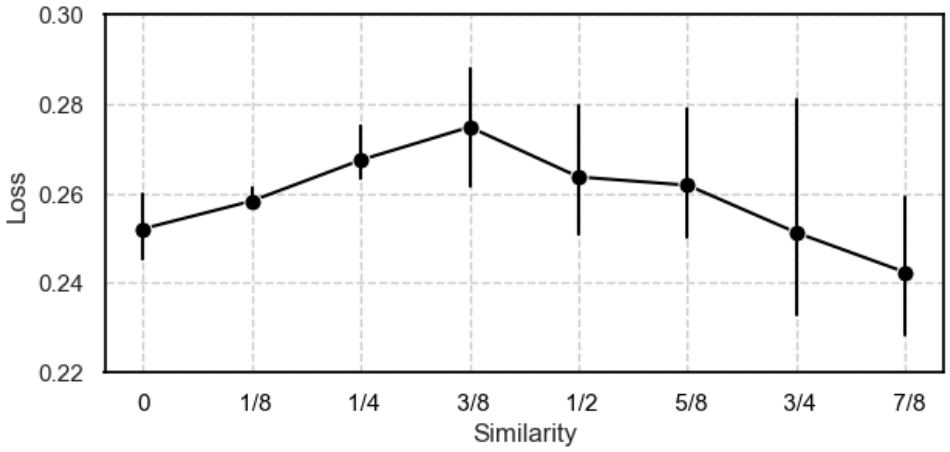

Figure B1 Results of the GPT-2 model.

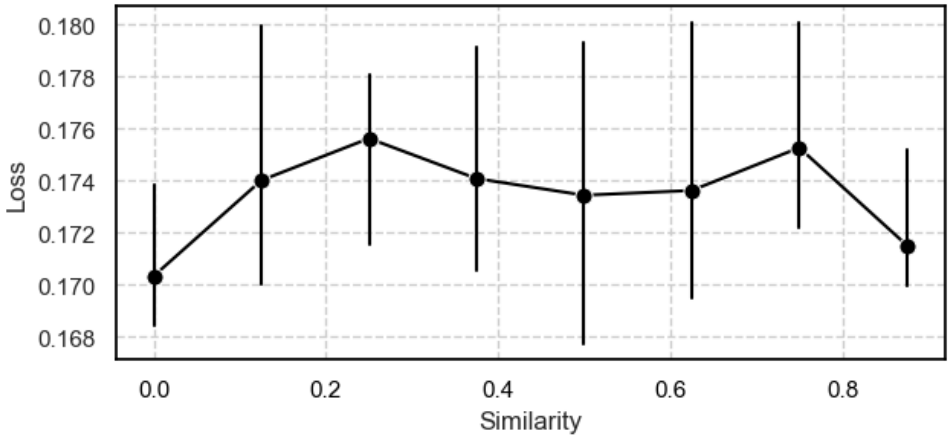

Figure B2 Results of the Llama 3.1 model.

Training set and test set. The training set consists of n_0 samples drawn from the base distribution and n_1 samples drawn from the another distribution of similarity r. The training set is fully shuffled before being batched and fed to the learning algorithm. The test set consists solely of the reference string of the base distribution c_0, which means the model is evaluated only on the prediction of each bit of c_0.

Model and training process. In this experiment, a GPT-2 model [5] with a vocabulary size of 3, 768 dimensions of word embeddings, 12 layers and 12 heads is selected. The model is loaded in mixed precision throughout the training and evaluation process. The model's context length is set to L, which is the same as the maximum length of samples. Samples with a length smaller than L are padded to the maximum length before training. During training, a batch size of 1024 is applied. The optimizer is an AdamW optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\varepsilon = 1\text{e}{-8}$ and a weight decay of 0.01. For the learning rate schedule, we employ a linear warmup followed by cosine decay schedule. Empirical tuning shows that setting the learning rate's initial value as $3\text{e}{-5}$, peak value as $3\text{e}{-4}$, end value as 0 and warmup steps as 1/10 of the full steps can guarantee the model's test loss converges. Note that we require the convergence of test loss to facilitate comparison, although Theorem A1 and Theorem A2 do not necessarily require the algorithm to be convergent.

Parameters. In the experiments of the GPT-2 model, we let L to be 64, δ to be 0.2, n_0 to be 100000 and n_1 to be 16384. Additionally, we vary the similarity r from 0 to 7/8 by 1/8. In the experiments of the Llama 3.1 model, we choose different parameters because the increased model scale causes fewer samples in a batch. These papameters are selected for convenience, and we believe our results still hold true for other sets of reasonable parameters.

We conduct the above experiments for 5 times to nullify the effect of randomness, as the generation of reference strings is random. For example, if the reference string of the distribution of similarity 1/2 is different from c_0 in the second half, the test loss should be significantly larger than the case that it is different from c_0 in the first half, where the model can distinguish two distributions easily.

Results and discussion The results of this set of experiments are shown in Figure B1 and Figure B2. Each bar represents the averaged loss of the model trained on a specific training set. And the error bars represent the maximun and minimum losses of the models during every repeatitions.

As illustrated by Figure B1, the model's test loss increases from 0 to 3/8 and then decrease from 3/8 to 7/8. As 3/8 is

Table C1 The Mistral 7B model's HumanEval score after fine-tuning on different datasets.

fine-tuning dataset	PIQA	Social IQA	Boolq	HellaSwag	GSM 8K	MATH	MBPP
HumanEval score (%)	26.83	25	28.05	27.44	26.22	28.66	29.27

Table C2 The Mistral 7B model's HumanEval score after fine-tuning on the MATH dataset with different prompt styles.

Index	Dataset format	HumanEval score (%)	MATH score (%)
(a)	Question: Solve 24 = 1601*c - 1605*c for c. Solution: -6	31.70	29.85
(b)	Question: Solve 24 = 1601*c - 1605*c for c. Answer: -6	29.88	23.88
(c)	Solve 24 = 1601*c - 1605*c for c. Solution: -6	27.44	9.70
(d)	Solve 24 = 1601*c - 1605*c for c. Answer: -6	25.61	21.64
(e)	Question: 'Solve 24 = 1601*c - 1605*c for c.' Solution: "-6"	27.44	2.98
(f)	Question: 'Solve 24 = 1601*c - 1605*c for c.' Answer: "-6"	29.27	13.43
(g)	'Solve 24 = 1601*c - 1605*c for c.' Solution: '-6'	10.98	0
(h)	'Solve 24 = 1601*c - 1605*c for c.' Answer: '-6'	12.20	0
(i)	def solve_linear_equation(): """ Solve 24 = 1601*c - 1605*c for c. """ return (-6)	14.63	9.70

close to but lower than 1/2, this result is in line with Theorem A2.

Another noteworthy observation from these experiments is that the losses exhibit considerable variability across multiple repetitions. Further investigation indicates that this phenomenon is due to the fact that the positions where a reference string c_i differs from c_0 exert a great influence on the trained model's loss, despite the similarity between c_i and c_0 remaining constant. In particular, whether the first few bits are different matters too much. This observation does not contradict Theorem A2, since Theorem A2 is valid in terms of expectation. However, this observation reveals the shortcoming of this type of simplification, namely that the definition of similarity may be good in terms of expectation, but it is not suitable for specific strings.

Appendix C Details of experiments on real-world data

Appendix C.1 Observing the performance drift phenomenon

To observe the performance drift phenomenon in real-world settings, in this section, we fine-tune one of the most recent transformer models, the Mistral 7B model [6], on different commonly-used datasets and observe its performance on the base task, namely the Python code generation task. More specifically, the base task is to complete a Python code test formatted in the same format as the HumanEval dataset [7]. This choice is made because this base task is relatively objective and easy to evaluate compared to other semantic tasks. For evaluation, the model is evaluated on the HumanEval dataset, which has 164 problems. A score commonly refered to as "pass@1" is used as the metric of the model's performance on the base task. This score represents the proportion of the 164 problems on which the model's generated code completion can pass the test.

In the fine-tuning process, the model is fine-tuned using lora [8, 9] for 60 steps. We use a batch size of 8, which means that the full dataset size for fine-tuning is 480, and we use an adamw 8bit optimizer. For the learning rate schedule, we employ a linear decay schedule with 5 steps of warm up that peaks at 2e−4.

The results are presented in Table C1. For comparison, the Mistral 7B model's "pass@1" score on the HumanEval dataset before fine-tuning is 28.05%. It is not surprising that fine-tuning on most of the common datasets did not significantly impact the model's performance on the base task. Here we argue that a single task or dataset can be viewed as a mixture of subpopulations. On average, the mixture may exhibit a small similarity to the base task. However, there is no guarantee that such a dataset does not contain some subpopulation with an intermediate similarity to the base task. We hypothesize

```
def remove_Occ(s,ch): exit
 assign """s    bool check)) function tooe"\ andev occurlists of dig given character from the string g
Points """
    formerge in left -=len(s)):o
       low ( "{[i] == .):",
           ram = s adjacentolution : i]In s[i + 1._Number
            break
    for i in range Nodelenins) - )),-ences,-)):sin
        ifusts[i] == ch):  last            s =|[0 : i] N s[i8 1:]uler
            break
    returnram ity</s>
```

Figure C1 An example from the 0.7-similarity dataset.

that it is quite possible that a dataset may contain subpopulations with an intermediate similarity to the base task, while their portion in the dataset may be relatively small. This explains the results presented in Table C1, that fine-tuning on some common datasets did not result in a significant degradation. However, over time, it is possible that significant degradation on the base task may occur.

Given the lack of significant performance drift observed in the above experiments, we further fine-tune the Mistral 7B model on the MATH dataset with different prompt styles. The prompt styles and the corresponding results are presented in Table C2.

By employing different prompt styles, we believe the similarity of the dataset to the base task varies. For instance, we believe that prompt style (a) yields the most dissimilar dataset, while prompt style (i) yields the most similar dataset. This is evidenced by the observation from Table C2 that prompt style (a) does not significantly alter the model's score, while prompt style (i) degrades the model's score to a lesser extent. By modifying prompt style (a), we believe that the dataset's similarity increases from (a) to (g). This, in turn, leads to a corresponding decrease in the model's score. In particular, the prompt style (g) yields the poorest score, 10.98%, which we believe represents a significant observation of the phenomenon of performance drift. These results support our theory.

Appendix C.2 fine-tuning models on constructed datasets of different similarities

To illustrate the idea that intermediate task similarity destroys a model's ability most severely, in this set of experiments, we fine-tune a base model on datasets that are deliberately constructed such that their similarities to a certain base task can be better controlled. After fine-tuning, the model is again evaluated on that base task. In our expectation, fine-tuning on datasets with intermediate similarity should make the model deteriorate the most.

Models and tasks. Two recent transformer models are used in this set of experiments, the Mistral 7B model [6] and the Llama3 8B model[1)]. We choose the same base task and evaluation score as in Section Appendix C.1, i.e., the base task is a Python code generation task and the evaluation score is the "pass@1" score on the HumanEval dataset.

How the datasets are constructed. We would like to construct datasets of any similarity r. However, due to the difficulty in defining task and task similarity in LLMs, we only employ an approximate method, which controls similarity in a rough manner. For the 1-similarity dataset, we reformat the MBPP dataset [10] in HumanEval's format and use it as the dataset of 1 similarity. We believe that since the MBPP dataset is also a Python code generation dataset, the distribution of the reformatted MBPP dataset should be very close to that of the base task. To generate the r-similarity dataset for $r < 1$, first we generate 8 strings of 2048 tokens randomly from commonly used tokens, which is all the tokens that appear in the MBPP dataset in this case. Subsequently, we modify the 1-similarity dataset using these 8 strings to generate the r-similarity dataset. For each sample s_1 in the 1-similarity dataset, a string is randomly selected from the 8 strings, denoted s_2. Then, each token in s_1 is substituted by the token of the same position in s_2 with a probability of $1 - r$. For instance, the ith token in s_1 could be substituted by the ith token in s_2. The result of substituting is then added to the r-similarity dataset. An example of samples in the 0.7-similarity dataset is shown in Figure C1. The pseudo code of this procedure is shown in Algorithm C1

Algorithm C1 Pseudo code for modifying a sample from the original dataset

Input: similarity r, original string s_1, artificial string s_2
Output: Modified string s

Initialize s to be an empty string
for $\mathrm{l} = 1$ to $\mathrm{len}(s_1)$ **do**
  Choose a random number uniformly from $[0, 1]$, denoted by rnd
  if $\mathrm{rnd} < r$ **then**
    Append s by $s_1(l)$
  else
    Append s by $s_2(l)$
  end if
end for

1) https://llama.meta.com/llama3

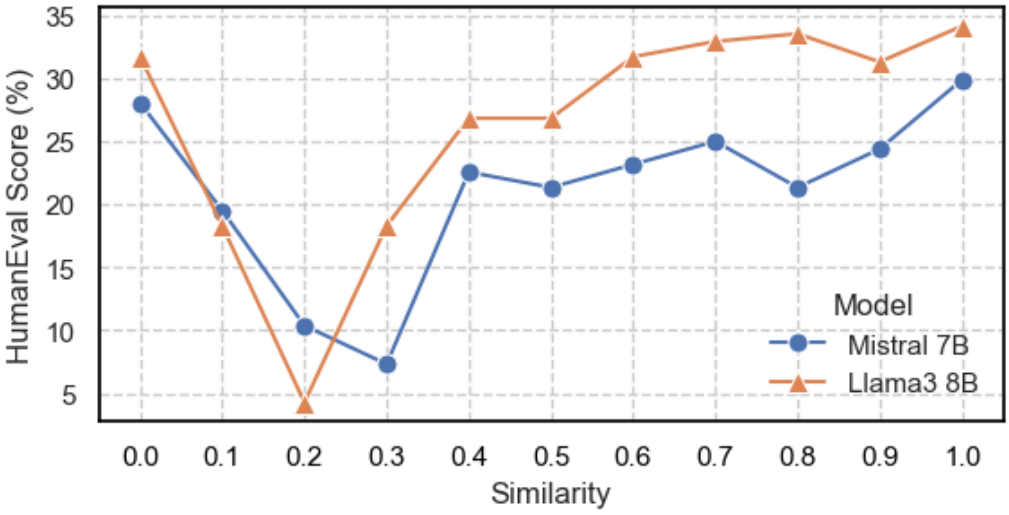

Figure C2 Results of the two models after fine-tuning.

fine-tuning. The fine-tuning parameters are the same as in Section Appendix C.1. For the learning rate schedule, we found that the Llama3 8B model (loaded in 4 bits) is much more vulnerable to learning rate. Only very small learning rate can improve the model's score after fine-tuning on the 1-similarity dataset, which in turn makes the results not significant. Consequently, we choose the same learning rate schedule across the fine-tuning process of the two models, which is a linear decay schedule with 5 steps of warm up that peaks at 2e−4.

Results and discussion After fine-tuning, the models' "pass@1" score is recorded as the metric of the models' ability on code generation task. The results are presented in Figure C2.

First we would like to note that fine-tuning on the 0-similarity dataset does not significantly impact the model's code generation ability, and fine-tuning on the 1-similarity dataset results in the highest score. The most detrimental impact on the model's ability occurs when similarity is lower than 0.5 but larger than 0. On the one hand, this result is consistent with the hypothesis that intermediate similarity impairs the model's ability to the worst. On the other hand, this result is in agreement with Theorem A2, which argues that an algorithm's averaged error rate is maximized when the input's similarity is lower than 1/2.

Another noteworthy observation is that the similarity when the model deteriorate the most is not near 1/2. For Mistral 7B, it is 0.3, and for Llama3 8B, it is 0.2. We argue that our definition of task similarity may be flawed and does not accurately reflect the true value. It can be stated with certainty that the 0-similarity dataset exhibits a similarity close to 0, and the 1-similarity dataset exhibits a similarity close to 1. With regard to the remaining datasets with similarities values between 0 and 1, our definition of similarity can only be regarded as an approximation. The ideal methodology for defining similarity and constructing datasets would be to mix the data distribution of the 0-similarity task and the 1-similarity task, i.e., a sample in the dataset is constructed by predicting the next token from an empty string using a mixture of distribution, until the string finishes. However, such a methodology is impratical and impossible, given that the data distribution itself is unknown to us.

Appendix C.3 Repeating the experiments with another base task, BoolQ

To enhance the persuasiveness of our findings, we also change the base task from the Python generation task to a reading comprehension task, evaluated by the BoolQ dataset [11], and repeat the above experiments. This base task is selected because the BoolQ dataset is also a relatively objective and easy to evaluate dataset. We evaluate the models' performance on this base task by calculating the proportion of the validation split of the BoolQ dataset on which the models can provide the correct response.

The dataset constructions is similar to that described in Section Appendix C.2. The 1-similarity dataset is the train split of the BoolQ dataset. Additionally, the 8 strings of 2048 tokens utilized in Section Appendix C.2 are reused. The r-similarity dataset for any $0 \leqslant r < 1$ is constructed by the 1-similarity dataset and these strings in the identical manner as in Section Appendix C.2.

Following the same fine-tuning process as outlined in Section Appendix C.1, the results of the models' evaluation scores are presented in Figure C3. It can be observed from this figure that the behavior of the two models is consistent with that observed in Section Appendix C.2, despite the change of the base task. The Mistral 7B model suffers the greatest decline in performance when the similarity is set to 0.3, while the Llama3 8B model suffers the greatest decline in performance when the similarity is set to 0.2.

As discussed Appendix A, the model's inherent ability to distinguish and separate different subpopulations is crucial for mitigating performance drift. The fact that the similarities leading to the worst performance for the two models are different, as shown in Figure C2 and Figure C3, can be viewed as an aspect of the models' ability in a sense. In the view of subpopulations, there are several possible explanations for this observation. One potential explanation is that the Mistral

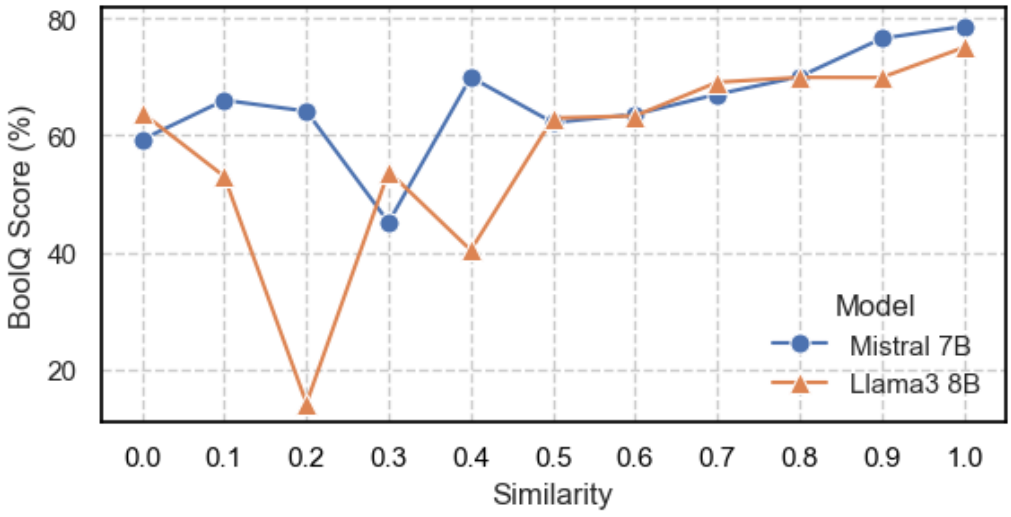

Figure C3 Results of the two models' scores after fine-tuning. The base task is changed to BoolQ.

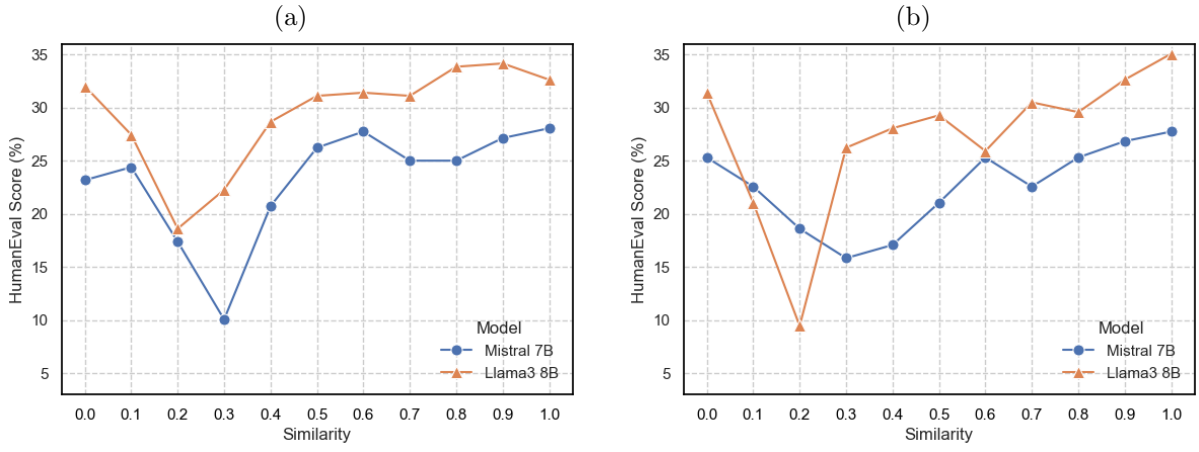

Figure C4 (a) Results of fine-tuning Mistral 7B with datasets of another prompt style; (b) Results of inferrence with 3-shot prompt after fine-tuning the Mistral 7B model.

7B model has a higher threshold for distinguishing different subpopulations. Another possibility is that the Mistral 7B model's prediction is less affected by outliers within the same subpopulation. It is also possible that both of these factors are at play.

Appendix C.4 Mitigating the performance drift phenomenon

As previously demonstrated in Section Appendix C.1, prompt style plays a significant role in performance drift. In this section, we attempt to investigate how the prompt can help mitigate performance drift.

Through further experiments, we found that prompt style matters in both training and inference. Here we use the base task of Python code generation. When the training set is not formatted in HumanEval's style but rather in another prompt style, the impact of fine-tuning on the model's performance on the base task is limited. For example, if the 1-similarity dataset is constructed by samples from the MBPP dataset in a question-answering format, the models' score on all the datasets of intermediate similarity will not deteriorate too much, as shown in Figure C4(a). Alternatively, if the models are prompted with in-context 3-shot prompt during inference, their scores will be more consistent and less degraded, as illustrated in Figure C4(b).

These two points provide us with insights into methods of mitigating performance drift during fine-tuning and inference. First, it is recommended to format the training set with a specific format to facilitate the model's ability to distinguish different tasks. Second, the use of in-context prompts during inference helps the model to recall the corresponding data distribution.

References

1 Gavin Brown, Mark Bun, Vitaly Feldman, et al. When is memorization of irrelevant training data necessary for high-accuracy learning? In Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, 2021. 123–132

2 Minki Kang, Seanie Lee, Jinheon Baek, et al. Knowledge-augmented reasoning distillation for small language models in knowledge-intensive tasks. In Advances in Neural Information Processing Systems, 2023. 48573–48602
3 Vitaly Feldman. Does learning require memorization? a short tale about a long tail. In Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, 2020. 954–959
4 Thomas M Cover. Elements of information theory. John Wiley & Sons, 1999.
5 Alec Radford, Jeffrey Wu, Rewon Child, et al. Language models are unsupervised multitask learners. OpenAI Blog, 2019, 1(8):9
6 Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, et al. Mistral 7b. arXiv preprint arXiv:2310.06825, 2023
7 Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021
8 Edward J Hu, yelong shen, Phillip Wallis, et al. LoRA: Low-rank adaptation of large language models. In International Conference on Learning Representations, 2022
9 Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, et al. Qlora: Efficient finetuning of quantized llms. In Advances in Neural Information Processing Systems, 2023, 10088–10115
10 Jacob Austin, Augustus Odena, Maxwell Nye, et al. Program synthesis with large language models. arXiv preprint arXiv:2108.07732, 2021
11 Christopher Clark, Kenton Lee, Ming-Wei Chang, et al. BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), 2019, 2924–2936