Everything Everywhere All at Once: LLMs can In-Context Learn Multiple Tasks in Superposition

Zheyang Xiong [1] **Ziyang Cai** [1] **John Cooper** [1] **Albert Ge** [1] **Vasilis Papageorgiou** [1] **Zack Sifakis** [1] **Angeliki Giannou** [1] **Ziqian Lin** [1] **Liu Yang** [1] **Saurabh Agarwal** [2] **Grigorios G Chrysos** [1] **Samet Oymak** [3] **Kangwook Lee** [1] **Dimitris Papailiopoulos** [1 4]

Abstract

Large Language Models (LLMs) have demonstrated remarkable in-context learning (ICL) capabilities. In this study, we explore a surprising phenomenon related to ICL: LLMs can perform multiple, computationally distinct ICL tasks simultaneously, during a single inference call, a capability we term "task superposition". We provide empirical evidence of this phenomenon across various LLM families and scales and show that this phenomenon emerges even if we train the model to in-context learn one task at a time. We offer theoretical explanations that this capability is well within the expressive power of transformers. We also explore how LLMs internally compose task vectors during superposition. Furthermore, we show that larger models can solve more ICL tasks in parallel, and better calibrate their output distribution. Our findings offer insights into the latent capabilities of LLMs, further substantiate the perspective of "LLMs as superposition of simulators", and raise questions about the mechanisms enabling simultaneous task execution.

1. Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities across various domains, with one of the most intriguing being in-context learning (ICL) (Brown et al., 2020; Xie et al., 2022a). ICL enables LLMs to perform tasks during inference without the need to fine-tune for that particular task, simply by providing a few examples within the input prompt. This ability has sparked significant interest in the research community, as it suggests that LLMs can adapt to novel tasks on-the-fly, using the capabilities that they acquired during pretraining, and the context provided.

While ICL has been extensively studied from both theoretical and empirical perspectives (Xie et al., 2022a; Agarwal et al., 2024), many aspects of its underlying mechanisms remain elusive. In this work, we study a surprising phenomenon related to ICL that, to the best of our knowledge, has not been thoroughly studied before: LLMs can perform multiple distinct ICL tasks simultaneously, in a single inference call, a capability we refer to as *"task superposition"*.

Our study suggests that pretrained autoregressive LLMs such as Llama (Touvron et al., 2023) or GPT-3.5[1] (Brown et al., 2020) display superposition of tasks purely *in-context*[2]. When presented with multiple in-context examples from different tasks, in the same prompt, the models can generate outputs that correspond to solutions for all these individual tasks. For instance, given examples of addition and translation, the model can concurrently produce correct answers for both tasks, as well as the composition of these tasks (e.g., the result of addition translated into another language).

Figure 1 illustrates this phenomenon. In Figure 1a (left), given in-context examples of addition in different languages and the query "91 + 83 →", the model generates probabilities for the correct sum in various languages, demonstrating its ability to perform addition and translation concurrently.

This discovery aligns and lends further support to the view of LLMs as superposition of simulators (Janus, 2022; Shanahan et al., 2023; Nardo, 2023) and the Bayesian perspective of ICL proposed by Xie et al. (2022a). While not a mathematically rigorous formulation, we can conceptualize the output of an LLM as a weighted sum of conditional probabilities across possible tasks:

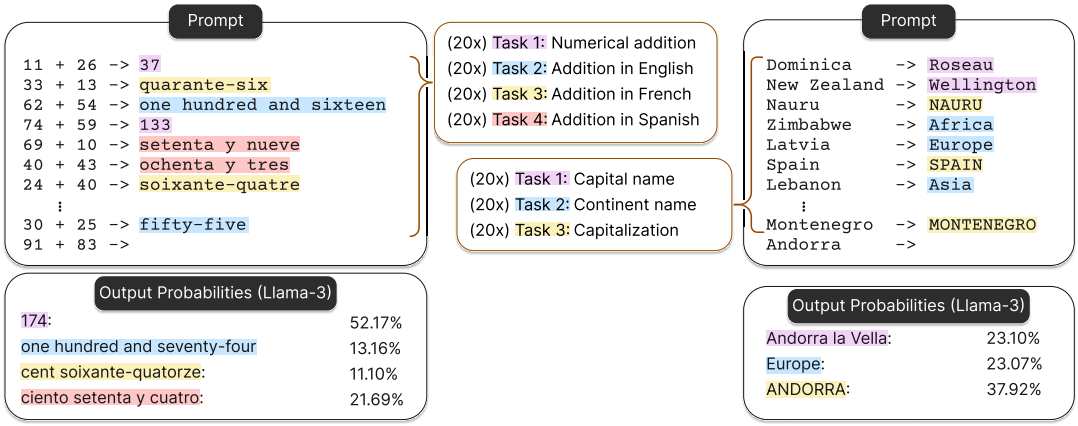
$$\mathbb{P}(\mathsf{output}|\mathsf{prompt}) \approx \sum_{\mathsf{task}} \mathbb{P}(\mathsf{output}|\mathsf{task}, \mathsf{prompt})\mathbb{P}(\mathsf{task}|\mathsf{prompt}).$$

[1]University of Wisconsin-Madison [2]University of Texas at Austin [3]University of Michigan [4]Microsoft Research. Correspondence to: Zheyang Xiong <zheyang@cs.wisc.edu>, Dimitris Papailiopoulos <dimitris@papail.io>.

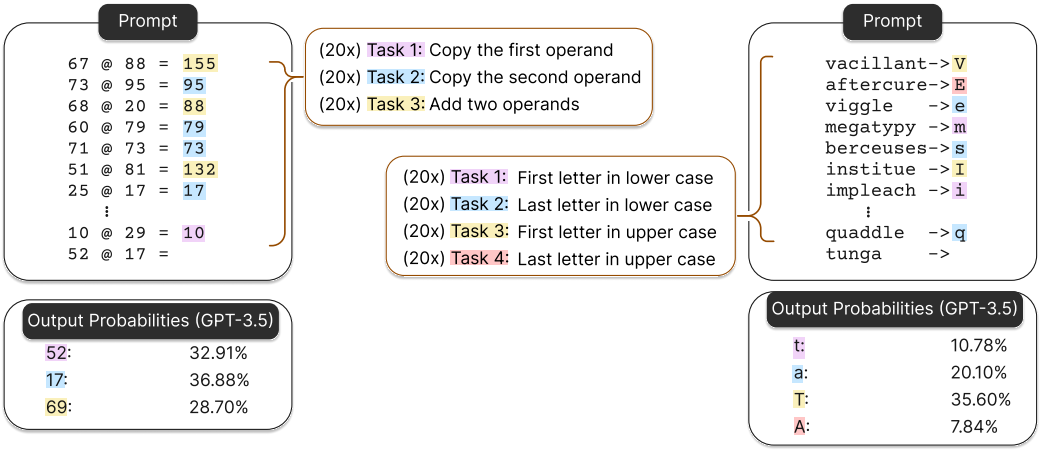
Proceedings of the 42nd International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

[1]In particular, `gpt-3.5-turbo-instruct`.

[2]For other definitions of superposition, please see Section 2.



(a) **(left)** Two-digit addition in a variety of languages. **(right)** Naming the capital of a given country name, naming the continent of a given country name or capitalizing the country name.



(b) **(left)** Tasks `copy(op1)`, `copy(op2)` and `op1+op2`. **(right)** First or last letter in upper or lower case[a].

[a]In the actual prompt we use space to separate each letter in a word so that each letter is a separate token after tokenization.

Figure 1: LLMs can perform task superposition. **(a)** Llama-3 70B and **(b)** GPT-3.5 Turbo are each presented with two sets of tasks. For each set of tasks, we show an example prompt such that all except the last row are in-context examples of one of the tasks and the last row is the query. We provide 20 in-context task examples for each task in the prompt with order randomized and provide the probabilities of outputs when correctly performing each task on the query.

In this conceptual model, $\mathbb{P}(\mathsf{output}|\mathsf{prompt})$ represents the probability distribution over possible outputs given the input prompt, a task can be thought of as a latent variable representing different capabilities the model might possess (e.g., arithmetic, translation, sentiment analysis), $\mathbb{P}(\mathsf{output}|\mathsf{task}, \mathsf{prompt})$ represents the output probability distribution if the model was specifically attempting to solve a single task, based on the test example in the prompt, and $\mathbb{P}(\mathsf{task}|\mathsf{prompt})$ represents the model's inferred probability that the prompt specifies a particular task.

While this mental model is a simplification of how an LLM operates, it provides an intuitive way to support the task superposition phenomenon we observe. Our findings lend support to the idea that LLMs can simultaneously maintain and utilize multiple task distributions, resulting in outputs that reflect a combination of relevant tasks.

Our Contributions: Our study makes several key contributions:

1. Through extensive empirical investigation and theoretical results, we demonstrate that task superposition is prevalent across various pretrained LLM families (GPT3.5, LLama-3, Qwen).
2. We empirically show that task superposition emerges as we train on one task at a time.
3. We provide a theoretical construction showing that Transformers models are indeed capable of task super-

position, and have the capacity to implement multiple tasks in parallel.

4. We explore how LLMs internally compose task vectors (Hendel et al., 2023) during superposition, and show how convex combinations of task vectors can reproduce the superposition effect.
5. We show that larger models can solve more tasks in parallel and more accurately reflect the distribution of in-context tasks.

We believe that our findings offer new insights into the latent capabilities of LLMs and raise questions about the mechanisms enabling simultaneous task execution. We believe this work sheds more light on the ICL capabilities of frontier language models, and offers a glimpse on potential applications of task superposition in practical settings.

2. Related Work

Theory and practice of in-context learning. There is rich literature which formalizes in-context learning under diverse definitions. For example, prior works study in-context learning through a Bayesian framework for task retrieval (Xie et al., 2022a; Panwar et al., 2023; Zhang et al., 2023), martingales (Falck et al., 2024), optimizers (Akyürek et al., 2023; Oswald et al., 2023; Dai et al., 2022) and more (Reddy, 2024; Olsson et al., 2022). Other works confirm the theoretical framing of in-context learning by using it to implement a variety of algorithms and methods (Zhou et al., 2023; Ahn et al., 2023; Giannou et al., 2023; Wu et al., 2024; Laskin et al., 2022; Zhou et al., 2022), or to approximate general-purpose computing machines (Giannou et al., 2023; Wei et al., 2022).

To bridge the gap between theory and practice, many works have used these theoretical insights to study in-context learning behaviors (Agarwal et al., 2024; Li et al., 2024; Choi & Li, 2024). Bai et al. (2023b) shows transformers are capable of in-context algorithm selection. We would like to note here that while one can think of superposition as an algorithm that outputs probability distribution that calibrates to in-context examples, the setting in Bai et al. (2023b) requires training on all such algorithm that calibrates the output distribution to in-context examples, which is likely not the case for pretrained LLMs since labels during LLMs training are in one-hot format. Therefore, while related, superposition cannot be explain by the framework from Bai et al. (2023b).

Other works study the factors that influence how well models can learn through context (Raventos et al., 2023; Chan et al., 2022; Wei et al., 2023; Lin & Lee, 2024; Min et al., 2022; Lyu et al., 2022; Lampinen et al., 2022; Dinh et al., 2022).

The development of new architectures such as state space models (Gu & Dao, 2023) has further motivated studying whether in-context learning is prevalent in alternative architectures such as Mamba (Park et al., 2024; Grazzi et al., 2024; Zeng et al., 2024) or in looped transformers (Yang et al., 2023).

Steering models through in-context learning has been a growing area of interest. Recent work has hypothesized that in-context learning can be encapsulated by a high-dimensional description of a task, which can be used to replace, (Hendel et al., 2023) compose (Todd et al., 2024) or augment (Liu et al., 2024) the latent states of a model, in order to alter its default behavior. Task vectors can be combined via arithmetic operations to solve a variety of tasks (Ilharco et al., 2023). Prior work has also been investigating the power of tokens in defining a task (Bai et al., 2024).

Other definitions of superposition. Our findings on superposition are inspired by notions of language models as multiverse generators (Reynolds & McDonell, 2021; moire, 2021). One consequence of LLMs as a superposition of tasks is that the outputs may collapse to unintended simulacra, a behavior known as the "Waluigi effect" (Nardo, 2023).

Superposition has been defined in various related contexts of learning models. Feature superposition (Elhage et al., 2022) refers to a neural network's ability to represent multiple learned concept in a single neuron. Though our discovery of task superposition describes the same abstract idea, we stress that it is distinct from feature superposition because task superposition is most apparent in the final output of a model.

Superposition is also described as a way to store multiple models in a single set of parameters (Cheung et al., 2019), processing multiple inputs simultaneously (Shen et al., 2024a; Murahari et al., 2022). In our work, we demonstrate task superposition directly as a result of language pre-training, without the necessity of additional adapters or decoding strategies.

3. Problem Setup

Notation. Let $\mathcal{X}$ be the input space, $\boldsymbol{x} \in \mathcal{X}$, and $\mathcal{Y}$ be the output space. We define a task as a function $\boldsymbol{g} : \mathcal{X} \to \mathcal{Y}$; we denote a task answer as $\boldsymbol{g}(\boldsymbol{x})$, and $(\boldsymbol{x}, \boldsymbol{g}(\boldsymbol{x}))$ as a task example of the task $\boldsymbol{g}$.

In-context learning setting. Let $\texttt{prompt} = \boldsymbol{s}_m \oplus \boldsymbol{x}^{(m+1)} = [\boldsymbol{x}^{(1)}, g(\boldsymbol{x}^{(1)}), ..., \boldsymbol{x}^{(m)}, \boldsymbol{g}(x^{(m)}), \boldsymbol{x}^{(m+1)}]$ be a prompt consisting of m task examples and a query $\boldsymbol{x}^{(m+1)}$. M is a model that predicts $M(\texttt{prompt}) \in \mathcal{Y}$. We say M learns the task $\boldsymbol{g}$ in-context if $M(\texttt{prompt}) = \boldsymbol{g}(x^{(m+1)})$, i.e., M learns $\boldsymbol{g}$ and performs the mapping on the query $\boldsymbol{x}^{(m+1)}$.

Let LM be a language model and let $\mathcal{V}$ be the vocabulary space (token space). Given a prompt, instead of outputing a $\boldsymbol{y} \in \mathcal{Y}$ directly, LM outputs a probability distribution over $\mathcal{Y}$, i.e., $\texttt{LM}(\texttt{prompt}) = [\mathbb{P}(\boldsymbol{y}_1|\texttt{prompt}), ..., \mathbb{P}(\boldsymbol{y}_{|\mathcal{Y}|}|\texttt{prompt})]$ for $\boldsymbol{y}_j \in \mathcal{Y}, j = 1, ..., |\mathcal{Y}|$ [3]. In previous works (Brown et al., 2020; Xie et al., 2022b) all task examples in a prompt are from a *single* task g; LM would put most of the probability mass on a single task answer.

We are the first to study a setting that task examples in a single prompt are from K different tasks, i.e., we have tasks $\boldsymbol{g}_1, ...\boldsymbol{g}_K$ which form $\texttt{prompt} = [\boldsymbol{x}^{(1)}, \boldsymbol{g}^{(1)}(\boldsymbol{x}^{(1)}), ..., \boldsymbol{x}^{(m)}, g^{(m)}(\boldsymbol{x}^{(m)}), \boldsymbol{x}^{(m+1)}]$ where $\boldsymbol{g}^{(j)} \in \{\boldsymbol{g}_1, ..., \boldsymbol{g}_K\}$. Since LM outputs a probability distribution over $\mathcal{Y}$, we calculate the probability mass it assigns to each task answer $\boldsymbol{g}_1(\boldsymbol{x}^{(m+1)}), ..., \boldsymbol{g}_K(\boldsymbol{x}^{(m+1)})$. In Figure 1 we depict $[\mathbb{P}(\boldsymbol{g}_1(\boldsymbol{x}^{(m+1)})|\texttt{prompt}), ..., \mathbb{P}(\boldsymbol{g}_K(\boldsymbol{x}^{(m+1)})|\texttt{prompt})]$ and find that LM assigns non-negligible probability on multiple task answers. We term this phenomenon *task superposition*.

4. LLMs Are Superpositions of Multiple In-Context Learners

In this section, we want to investigate if existing pre-trained models exhibit superposition of multiple tasks and whether this phenomenon is common.

> **Finding 1:** *LLMs can in-context learn multiple tasks in superposition when provided with prompts of a mixture of task examples.*

We consider four different settings of task mixtures.

1. Numerical addition and addition in English, French or Spanish ($K = 4$). Example prompt is shown in Figure 1a (left).
2. Given a name of a country, name the capital, continent or capitalize the country name ($K = 3$). Example prompt is shown in Figure 1a (right).
3. Given input "{op1}@{op2}", copy op1, op2 or add op1 and op2 ($K = 3$). Example prompt is shown in Figure 1b (left).
4. Given a word (with each letter separated by a space), output first letter or last letter in lower or upper cases ($K = 4$). Example prompt is shown in Figure 1b (right).

[3] In practice LM outputs a distribution over $\mathcal{V}$ but for simplicity here we assume inputs and outputs are single-tokens, i.e., $\mathcal{X} = \mathcal{V} = \mathcal{Y}$, and it naturally generalizes to the case when outputs are multi-tokens using a probability chain rule.

We provide GPT-3.5 (Brown et al., 2020), Llama-3 70B (AI@Meta, 2024) and Qwen-2.5 72B (Bai et al., 2023a) with prompts of uniform mixture of tasks (each task has 20 examples in the prompt ordered randomly). For each prompt consisting of in-context task examples (e.g., "$11 + 26 \rightarrow 37$" for the first task in the first setting) and a query (e.g., "$91+83 \rightarrow$"), we calculate the probabilities of outputs when correctly performing each task on the query and plot the distribution of probabilities for each task in Figure 2. Details on calculating the probabilities is in Appendix C.

Figure 2 reveals that in all four sets of tasks, all models have non-negligible median values of probabilities for at least two tasks. This indicates that the models can in-context learn multiple tasks in superposition when provided with prompts of a mixture of task examples.

We can also observe that, even though every task in a prompt has an equal number of in-context examples (20 examples), LLMs do not calibrate their output distribution perfectly with the in-context task example distribution and they still have bias on what task to perform. For example, Figure 2a(left) shows that Llama-3 70B prefers performing numerical addition over addition in other languages, Qwen-2.5 72B is not good at addition in Spanish while GPT-3.5 does not have a strong preference over a single task. On the other hand, in Figure 2a(right) GPT-3.5 has a strong preference over the capital task.

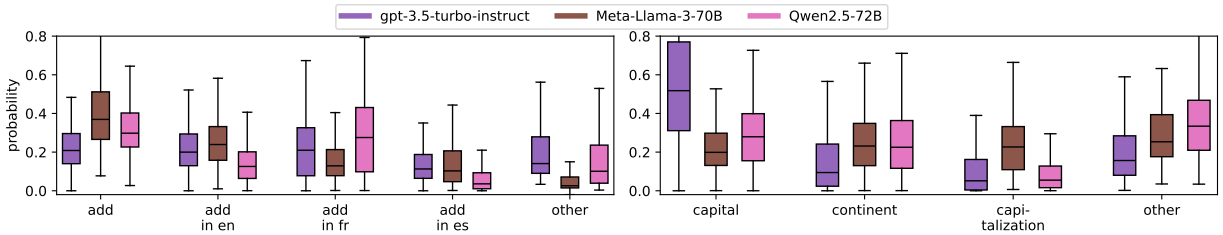
Additionally, some tasks are "harder" than other tasks. For example, in Figure 2b(right), all models assign low probabilities for task answers of last_letter_cap. The category other has relatively high values for all models, indicating a high noise when prompted with in-context examples of this setting. In contrast, in Figure 2b(left), category other has very small values, indicating that all models most of the time would correctly assign the output probabilities to the correct answers.

In Appendix B, we further show that, within the same LLM family, larger models can solve more tasks in parallel and better calibrate the output distribution to the in-context task examples distribution.

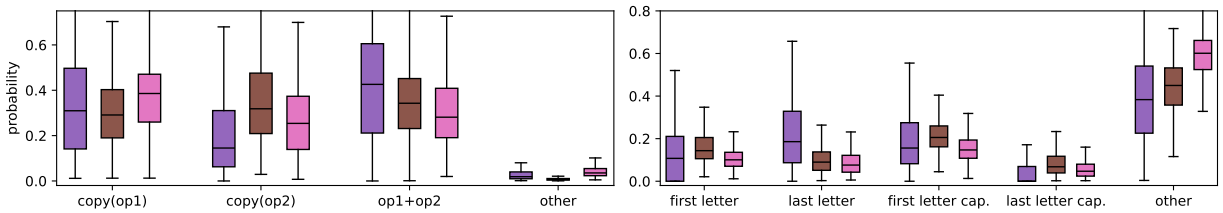
5. Task Superposition in Models Trained from Scratch

In Section 4 we investigated task superposition in pre-trained LLMs at inference time. We further investigate how task superposition emerges in LLMs during training. Specifically, if we train the model to in-context learn one task at a time, can it perform task superposition when provided with prompts containing examples of multiple tasks?

To answer this question, we train a small GPT-2 model (6 heads, 6 layers, ∼14million param-



(a) **(left)** Addition in original numerical form and in different languages as in Figure 1a (left); **(right)** Capital name, continent name and capitalization as in Figure 1a (right)



(b) **(left)** `copy(op1)`, `copy(op2)`, and `op1+op2` as in Figure 1b (left); **(right)** First or last letter in upper or lower cases as in Figure 1b (right).

Figure 2: Distributions (0/25/50/75/100-percentiles) of probabilities for correct outputs of each task. For every set of tasks, we tested with 100 prompts and for each prompt, every task has 20 random in-context task examples with order randomized like in Figure 1. Category `other` is the sum of probabilities of all other outputs. Gray dashed line in each figure is the ideal probability if we assume the model perfectly calibrates its output distribution to the distribution of in-context task examples. With uniform distribution of task examples, the dashed lines are at 0.25 (4 tasks setting) and 0.33 (3 tasks setting).

eters) (Radford et al., 2019) to learn a family of retrieval tasks. The input has the form "{ch1}{ch2}{ch3}{ch4}{ch5}{ch6}{ch7}{ch8}→" where `ch1`, ..., `ch8` are distinct single characters. We consider 8 retrieval tasks – `ret1`, ..., `ret8` – where `ret1` is to output `ch1` and so on. The model is trained to in-context learn one task (retrieve one of {`ch1`, ..., `ch8`}) at a time in training. Namely, during training, the model is only provided with text data such that each prompt only contains in-context examples of a single randomly chosen task (and different prompts can correspond to different tasks).

Concretely, for each sample, we randomly select task $t \in \{\texttt{ret1}, ..., \texttt{ret8}\}$ and inputs $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(m)}$, where each $\boldsymbol{x}^{(j)}$ is an eight-character long string. We train the model $\texttt{LM}_\theta$ parametrized by θ using ICL training by minimizing the following objective:

$$\min_\theta \mathbf{E}_{\boldsymbol{s}} \left(\frac{1}{m-1} \sum_{j=1}^{m-1} \text{CE}(\texttt{LM}_\theta(\boldsymbol{s}_j \oplus \boldsymbol{x}^{(j+1)}), \boldsymbol{g}_t(\boldsymbol{x}^{(j+1)})) \right), \tag{1}$$

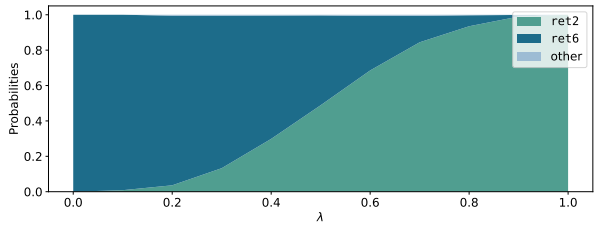
where CE is the cross-entropy loss and $\boldsymbol{s}_j \oplus \boldsymbol{x}^{(j+1)} \equiv [\boldsymbol{x}^{(1)}, \boldsymbol{g}_t(\boldsymbol{x}^{(1)}), ..., \boldsymbol{x}^{(j)}, \boldsymbol{g}_t(\boldsymbol{x}^{(j)}), \boldsymbol{x}^{(j+1)}]$.

After training, we provide the model with prompts containing in-context examples of two tasks (in particular, we choose `ret2` and `ret6`) and see if the model performs task superposition. We vary the proportion of in-context examples of two tasks and plot the output distributions in Figure 3a.

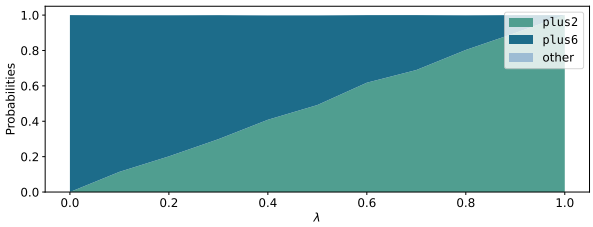
Similarly, we consider a second setting involving 10 tasks. Given a two digit integer input `num`, task `plus0` outputs `num`, task `plus1` outputs `num` + 1 and so on, up to task `plus9`. The model is trained to in-context learn one of `plus0`,..., `plus9` at a time, following the procedure above. During inference time, the model is tested with prompts containing a mixture of in-context examples from tasks `plus2` and `plus6`. We vary the mixture ratio and show the output distributions in Figure 3b.

Finding 2: *Transformers can in-context learn multiple tasks in superposition even if trained to in-context learn one task at a time.*

Remarkably, from Figure 3a and 3b, GPT-2 trained from scratch to in-context learn one task at a time can generalize to simultaneously performing multiple tasks and calibrate the output probabilities according to the in-context task example distribution when provided with a mixture of in-context examples. For example, in Figure 3a at the mixture ratio $\lambda = 0.5$, meaning that 50 percent of the examples in the prompt is from task `ret2` and the other 50% comes from task `ret6`, we can see the output probabilities for task



(a) Trained on retrieval tasks and tested on prompts with mixture of in-context examples of `ret2` and `ret6`.



(b) Trained on addition tasks and tested on prompts with mixture of in-context examples of `plus2` and `plus6`.

Figure 3: We consider two different settings of tasks: (a) given an eight-character length string as input, consider `ret1`, ..., `ret8` where `ret1` is to retrieve the first character and so on; and (b) given a two-digit integer as an input, consider `plus0`, ..., `plus9` where `plus0` is to add 0 on the input and so on. After training, for each setting, we select two tasks ad we provide the model with prompts containing in-context examples from these two tasks and vary the mixture ratio λ such that the in-context task example distribution for two tasks is $[\lambda, 1-\lambda]$. We plot λ on x-axis and the output probabilities of task answers for each task on y-axis.

answers of `ret2` and `ret6` being roughly $[0.5, 0.5]$. We can observe similar behavior in Figure 3b.

6. Transformers Have the Capacity to Perform Task Superposition

In this section, we explore whether Transformers have the inherent expressivity to perform multiple tasks in superposition with a single inference call. To this end, we provide a theoretical construction of a Transformer which, given the ability to implement multiple tasks, performs task superposition depending on the examples given in-context.

Theorem 1. *A seven layer transformer with embedding dimension $\mathcal{O}(d + \log(mn))$ with K heads per attention layer can perform K tasks on vectors of dimension d in superposition, with weighting based on m different in-context examples each of length n .*

The proof of Theorem 1 is provided in Appendix E.4. Note that while this does not guarantee that training a Transformer will actually find these parameters, it does indicate that Transformers are expressive enough to perform task superposition at test time. Below we outline the main ideas used in the proof.

Prediction based on multiple tasks. Assume that we are given m in-context samples $(\boldsymbol{x}_1^{(j)}, \ldots, \boldsymbol{x}_{n-2}^{(j)}, \text{'='}, \boldsymbol{y}^{(j)})_{j=1}^m$ where '=' represents a specific value used only for preceding the label, and a set of k different Transformers TF_i which can implement the T different desired tasks, where each deterministic task is denoted as $\boldsymbol{g}_i(\boldsymbol{x}^{(j)})$ with $i \in [k]$ and $j \in [m]$, *i.e.* $\boldsymbol{y}^{(j)} = \boldsymbol{g}_i(\boldsymbol{x}^{(j)})$ for some task i dependent on sample j. Using the weights of each TF_i, we can compute outputs of the following form:

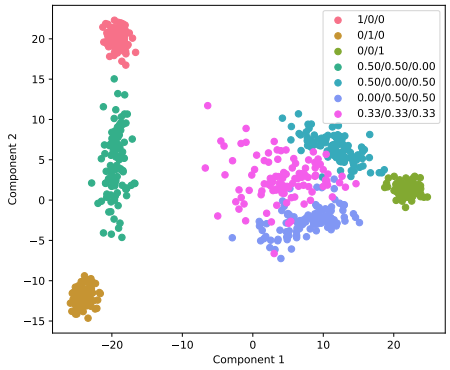
$$\begin{bmatrix} \ldots & \boldsymbol{x}_1^{(j)} & \ldots & \boldsymbol{x}_{n-2}^{(j)} & = & \boldsymbol{y}^{(j)} & \ldots \\ \ldots & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \ldots \\ & \vdots & & \vdots & \vdots & \vdots & \\ \ldots & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \ldots \end{bmatrix} \rightarrow \begin{bmatrix} \ldots & \boldsymbol{x}_1^{(j)} & \ldots & \boldsymbol{x}_{n-2}^{(j)} & = & \boldsymbol{y}^{(j)} & \ldots \\ \ldots & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} & \|\boldsymbol{g}_1(\boldsymbol{x}^{(j)}) - \boldsymbol{y}^{(j)}\|_1 & \ldots \\ & \vdots & & \vdots & \vdots & \vdots & \\ \ldots & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} & \|\boldsymbol{g}_T(\boldsymbol{x}^{(j)}) - \boldsymbol{y}^{(j)}\|_1 & \ldots \end{bmatrix}$$

We use the l_1 norm to aggregate the prediction, in case that the task is multi-dimensional. These differences are used to identify tasks, as $\|\boldsymbol{g}_i(\boldsymbol{x}^{(j)}) - \boldsymbol{y}^{(j)}\|_1 \approx 0$ for $\boldsymbol{y}^{(j)}$ coming from task i. Different heads at each layer in the model are used to execute each of the tasks in parallel using the weights from TF_i. In Appendix E we construct tasks where an arbitrary function $\boldsymbol{g}_i(\boldsymbol{x}_l^{(j)})$ is implemented using ReLUs for some fixed l that is task-specific.

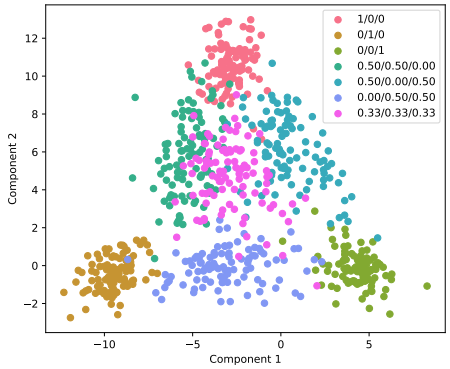
Creating task identifiers. Having the differences between the implemented function and the label, we first use the ReLUs to clean up the vectors $\boldsymbol{v}_k$ so that only the positions in each vector that are associated with a task are maintained and the rest are set to 1[4]. We thus create the vectors $(\boldsymbol{v}'_k)_i = \|\boldsymbol{g}_k(\boldsymbol{x}_{1:l-1}^{(j)}) - \boldsymbol{x}_l^{(j)}\|_1$ and $(\boldsymbol{v}'_*)_* = 1$ otherwise. Now we use ReLUs to threshold and create an indicator vectors $\mathbf{1}_{\{\|\boldsymbol{g}_k(\boldsymbol{x}_{1:l-1}^{(j)}) - \boldsymbol{x}_l^{(j)}\|_1 \approx 0\}}$ which identify the task, *i.e.*,these are task identifiers. Notice that if the task is correctly predicted then the difference should be close to 0 (up to some error), while if the task is not identified the corresponding value would not be 0; the rest of the rows would be 1. We have created one vector for each task, which has 1 in the position of the corresponding task if the task was identified in the context.

Averaging and task superposition. As a last step, we average all the task identifiers and place the result in the last column, in which the next prediction will happen. We then use the averaged task identifier to weight the prediction of each task based on it, as in task superposition. If the task has been identified multiple times in the context, it would be assigned a higher weight/probability.

[4]This step is not mandatory, but it ensures that we have no values over which we have no control. We leave as future work an error analysis on how these values could affect the task identifiers.



(a) `copy(op1)` / `copy(op2)` / `op1+op2`



(b) `to_fr` / `to_de` / `to_it`

Figure 4: Task vectors of Llama-3 8B projected onto two axes chosen by LDA for two sets of tasks: **(a)** `copy(op1)`, `copy(op2)` and `op1+op2` and **(b)** `to_fr`, `to_de` and `to_it`. For tasks t_1, t_2, t_3, we use "$\mathbb{P}(t_1)/\mathbb{P}(t_2)/\mathbb{P}(t_3)$" to denote different levels of task mixtures, e.g., "0.50/0.50/0.00" represents the case where the in-context task examples are 50% t_1, 50% t_2 and 0% t_3.

7. Task Superposition Through the Lens of Task Vectors

While in Section 6 we provide an existential result by constructing a Transformer that performs task superposition and shows that task superposition is well within the expressive power of Transformers, we would like to further investigate how task superposition manifest in pretrained LLMs internally. In this section we explore the underlying mechanisms that LLMs employ during task superposition. In particular, we focus our empirical study on *task vectors* (Hendel et al., 2023) where the detailed implementation is in Appendix D. Task vectors are vectors in the embedding space and are found to encode the algorithm that a model internally implements to solve a task given in-context demonstrations.

We want to investigate if there is any relation between the task vectors of each individual task and the task vectors of a mixture of task examples in the prompt. To this end, we consider two sets of tasks:

(a) `copy(op1)`, `copy(op2)` and `op1+op2` as in Figure 1b (left).

(b) Given a two-digit integer, task `to_fr` translates it to French, task `to_de` translates it to German and task `to_it` translates it to Italian.

For each set of tasks, we collect the task vectors for each individual task and task vectors extracted from prompts that contain examples of different tasks. In Figure 4, we project task vectors along two axes chosen by linear discriminant analysis (LDA).

Finding 3: *LLMs internally combine task vectors during task superposition.*

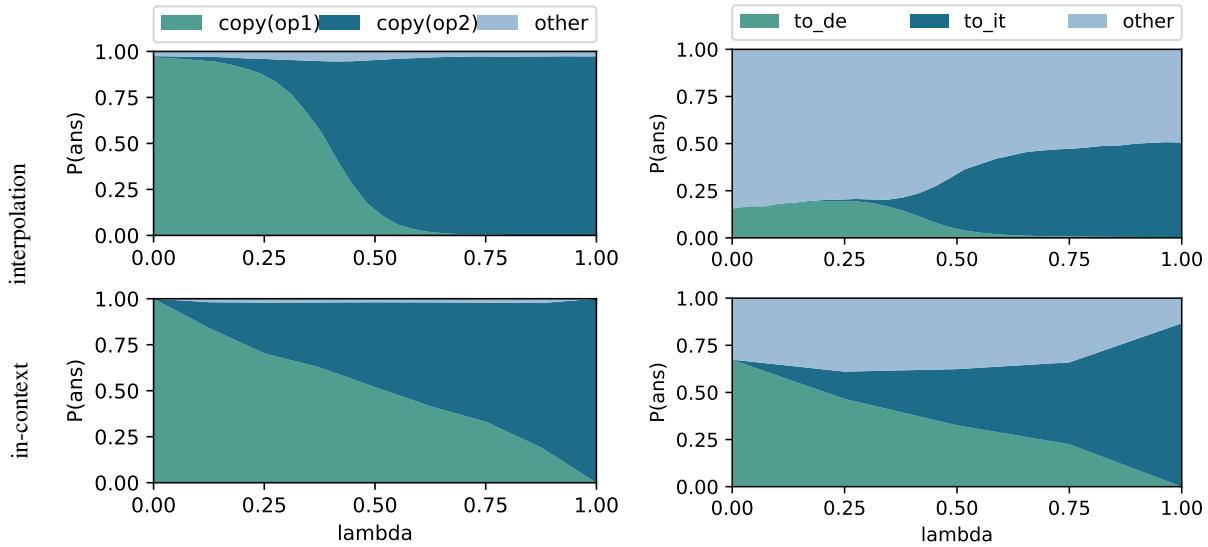
Interestingly, we observe that the locations of task vectors of a mixture of tasks strongly correlate with the locations of task vectors for each individual task and the in-context task example distribution (the mixture ratio for examples of different tasks). For example, if the prompt includes an equal number of in-context examples from each task, the task vectors are roughly centered in the middle; if the prompt only contains in-context examples of two tasks, then the task vectors roughly lie on the connecting line between task vectors of two individual tasks. We argue that this observation is indicative of the fact that, when prompted with a mixture of in-context task examples, LLMs internally combine task vectors.

As we observe signs that LLMs internally compose task vectors, we want to further investigate whether we can reproduce the task superposition phenomenon by patching in a convex combination of task vectors. For example, for tasks `copy(op1)` and `copy(op2)`, we first extract the corresponding task vectors $\mathrm{V}_{\texttt{copy(op1)}}$ and $\mathrm{V}_{\texttt{copy(op2)}}$ on Llama-3 8B using the method described in Appendix D. We then make a convex combination of the two task vectors with parameter λ that controls the ratio:

$$\mathrm{V}_{\text{interpolate},\lambda} = \lambda \cdot \mathrm{V}_{\texttt{copy(op1)}} + (1-\lambda) \cdot \mathrm{V}_{\texttt{copy(op2)}}.$$

Finding 4: *Convex combinations of task vectors produce task superposition.*

For a new query (in this scenario in the form "`{op1}@{op2}=`"), we patch the vector $\mathrm{V}_{\text{interpolate},\lambda}$ into the model at the task vector layer. We calculate the model output probabilities that correspond to each task while we vary λ. For each λ, we use 100 different queries and plot the average probabilities in the top row of Figure 5. As a comparison, in the bottom rows of Figure 5, we plot the corresponding output probabilities when providing the models



(a) Tasks: `copy(op1)` and `copy(op2)`

(b) Tasks: translate `to_de` and `to_it`

Figure 5: On Llama-3 8B, we vary the proportion, λ, between two tasks and observe how the output probabilities for the correct answers change. The proportion λ is varied in two ways: (1) in the top row, we plot the output from patching in a convex combination of task vectors for two tasks. (2) in the bottom row, we plot the output from a mixed proportion of in-context examples for the two tasks. Subplot (a) shows the output probabilities from mixing two copy tasks and (b) shows the probabilities from mixing two translate tasks.

with prompts containing mixture of task examples where the mixture ratio is controlled by λ.

In top row of Figure 5, we observe that patching convex combinations of task vectors into the model produces task superposition. We would also like to point out that in Figure 5b, although irrelevant outputs sum up to a large probability, the task answers for two tasks `to_de` and `to_it` in most cases will still be the top-2 answers.

Comparing the top rows and the bottom rows, we can see that top rows (the scenario of interpolating task vectors of individual tasks) have larger probabilities of irrelevant output (category `other`). Task vector interpolation also produces less of a linear relationship between λ and the output probabilities. This shows that while convex combinations of task vectors are sufficient for producing task superposition, this does not fully explain task superposition. We leave it to future work to investigate other mechanistic explanations of task superposition.

8. Limitations and Future Directions

One limitation of our work is the current gap between the demonstrated capability of LLMs to perform task superposition and its practical application in real-world scenarios. While we have shown that LLMs possess the capacity to execute multiple tasks simultaneously, conventional decoding algorithms are not equipped to fully leverage this capability. This limitation stems from what we term "generation collapse," a phenomenon where, after the first token is generated, the model tends to converge on predicting tokens for a single task, effectively negating its ability for multi-task execution.

This collapse presents a substantial challenge in harnessing the full power of task superposition. It highlights a critical area for future research: developing decoding strategies that can maintain the model's multi-task state throughout the generation process. Recent work by Shen et al. (2024b) offers some hope that this direction may be fruitful, by proposing a "superposed decoding" algorithm. Their method efficiently generates multiple streams of tokens from a single inference pass by utilizing superposed token embeddings. While this approach represents a significant step forward, it also highlights the potential for further innovation in this area.

9. Conclusion

We report on the discovery of task superposition, which is the ability of LLMs to simultaneously solve distinct tasks from in-context examples. Task superposition is present in a variety of pretrained models, and becomes more accurate at predicting the distribution of tasks as the model size increases (Appendix B). We also find evidence that while displaying task superposition, models internally mix the task vectors of each individual task. We hope that our findings will contribute to understanding in-context learning mechanisms and enhance our knowledge of LLMs overall.

Acknowledgment

K.L. is supported by NSF Award DMS-2023239, NSF CAREER Award CCF-2339978, Amazon Research Award, and a grant from FuriosaAI. D.P. is supported by NSF Medium award No. 2403074, ONR Grant No. N00014-21-1-2806 and No. N00014-23-1-2848.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Agarwal, R., Singh, A., Zhang, L. M., Bohnet, B., Chan, S., Anand, A., Abbas, Z., Nova, A., Co-Reyes, J. D., Chu, E., Behbahani, F., Faust, A., and Larochelle, H. Many-shot in-context learning, 2024.

Ahn, K., Cheng, X., Daneshmand, H., and Sra, S. Transformers learn to implement preconditioned gradient descent for in-context learning. *arXiv preprint arXiv:2306.00297*, 2023.

AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

Akyürek, E., Schuurmans, D., Andreas, J., Ma, T., and Zhou, D. What learning algorithm is in-context learning? Investigations with linear models, May 2023. URL http://arxiv.org/abs/2211.15661. arXiv:2211.15661 [cs].

Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., Hui, B., Ji, L., Li, M., Lin, J., Lin, R., Liu, D., Liu, G., Lu, C., Lu, K., Ma, J., Men, R., Ren, X., Ren, X., Tan, C., Tan, S., Tu, J., Wang, P., Wang, S., Wang, W., Wu, S., Xu, B., Xu, J., Yang, A., Yang, H., Yang, J., Yang, S., Yao, Y., Yu, B., Yuan, H., Yuan, Z., Zhang, J., Zhang, X., Zhang, Y., Zhang, Z., Zhou, C., Zhou, J., Zhou, X., and Zhu, T. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023a.

Bai, Y., Chen, F., Wang, H., Xiong, C., and Mei, S. Transformers as statisticians: Provable in-context learning with in-context algorithm selection, 2023b.

Bai, Y., Huang, H., Piano, C. S.-D., Rondeau, M.-A., Chen, S., Gao, Y., and Cheung, J. C. K. Identifying and analyzing task-encoding tokens in large language models. (arXiv:2401.11323), February 2024. doi: 10.48550/arXiv.2401.11323. URL http://arxiv.org/abs/2401.11323. arXiv:2401.11323 [cs].

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Chan, S., Santoro, A., Lampinen, A., Wang, J., Singh, A., Richemond, P., McClelland, J., and Hill, F. Data distributional properties drive emergent in-context learning in transformers. *Advances in Neural Information Processing Systems*, 35:18878–18891, 2022.

Cheung, B., Terekhov, A., Chen, Y., Agrawal, P., and Olshausen, B. Superposition of many models into one, June 2019. URL http://arxiv.org/abs/1902.05522. arXiv:1902.05522 [cs].

Choi, H. K. and Li, Y. Picle: Eliciting diverse behaviors from large language models with persona in-context learning. (arXiv:2405.02501), May 2024. doi: 10.48550/arXiv.2405.02501. URL http://arxiv.org/abs/2405.02501. arXiv:2405.02501 [cs].

Dai, D., Sun, Y., Dong, L., Hao, Y., Sui, Z., and Wei, F. Why can gpt learn in-context? language models secretly perform gradient descent as meta optimizers. *arXiv preprint arXiv:2212.10559*, 2022.

Dinh, T., Zeng, Y., Zhang, R., Lin, Z., Rajput, S., Gira, M., Sohn, J.-y., Papailiopoulos, D., and Lee, K. Lift: Language-interfaced fine-tuning for non-language machine learning tasks. *arXiv preprint arXiv:2206.06565*, 2022.

Elhage, N., Hume, T., Olsson, C., Schiefer, N., Henighan, T., Kravec, S., Hatfield-Dodds, Z., Lasenby, R., Drain, D., Chen, C., Grosse, R., McCandlish, S., Kaplan, J., Amodei, D., Wattenberg, M., and Olah, C. Toy Models of Superposition, September 2022. URL http://arxiv.org/abs/2209.10652. arXiv:2209.10652 [cs].

Falck, F., Wang, Z., and Holmes, C. Is in-context learning in large language models bayesian? a martingale perspective. (arXiv:2406.00793), June 2024. URL http://arxiv.org/abs/2406.00793. arXiv:2406.00793 [cs, stat].

Giannou, A., Rajput, S., Sohn, J.-y., Lee, K., Lee, J. D., and Papailiopoulos, D. Looped transformers as programmable computers, 2023.

Grazzi, R., Siems, J., Schrodi, S., Brox, T., and Hutter, F. Is mamba capable of in-context learning? (arXiv:2402.03170), April 2024. doi: 10.48550/arXiv.2402.03170. URL http://arxiv.org/abs/2402.03170. arXiv:2402.03170 [cs].

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. (arXiv:2312.00752), December 2023. doi: 10.48550/arXiv.2312.00752. URL http://arxiv.org/abs/2312.00752. arXiv:2312.00752 [cs].

Hendel, R., Geva, M., and Globerson, A. In-Context Learning Creates Task Vectors, October 2023. URL http://arxiv.org/abs/2310.15916. arXiv:2310.15916 [cs].

Ilharco, G., Ribeiro, M. T., Wortsman, M., Gururangan, S., Schmidt, L., Hajishirzi, H., and Farhadi, A. Editing Models with Task Arithmetic, March 2023. URL http://arxiv.org/abs/2212.04089. arXiv:2212.04089 [cs].

Janus. Simulators, 2022. URL https://www.lesswrong.com/posts/vJFdjigzmcXMhNTsx/.

Lampinen, A. K., Dasgupta, I., Chan, S. C., Matthewson, K., Tessler, M. H., Creswell, A., McClelland, J. L., Wang, J. X., and Hill, F. Can language models learn from explanations in context? *arXiv preprint arXiv:2204.02329*, 2022.

Laskin, M., Wang, L., Oh, J., Parisotto, E., Spencer, S., Steigerwald, R., Strouse, D., Hansen, S., Filos, A., Brooks, E., et al. In-context reinforcement learning with algorithm distillation. *arXiv preprint arXiv:2210.14215*, 2022.

Li, T., Zhang, G., Do, Q. D., Yue, X., and Chen, W. Long-context llms struggle with long in-context learning. (arXiv:2404.02060), April 2024. doi: 10.48550/arXiv.2404.02060. URL http://arxiv.org/abs/2404.02060. arXiv:2404.02060 [cs].

Lin, Z. and Lee, K. Dual Operating Modes of In-Context Learning, February 2024. URL http://arxiv.org/abs/2402.18819. arXiv:2402.18819 [cs].

Liu, S., Ye, H., Xing, L., and Zou, J. In-context vectors: Making in context learning more effective and controllable through latent space steering. (arXiv:2311.06668), February 2024. doi: 10.48550/arXiv.2311.06668. URL http://arxiv.org/abs/2311.06668. arXiv:2311.06668 [cs].

Lyu, X., Min, S., Beltagy, I., Zettlemoyer, L., and Hajishirzi, H. Z-icl: Zero-shot in-context learning with pseudo-demonstrations. *arXiv preprint arXiv:2212.09865*, 2022.

Min, S., Lyu, X., Holtzman, A., Artetxe, M., Lewis, M., Hajishirzi, H., and Zettlemoyer, L. Rethinking the role of demonstrations: What makes in-context learning work? (arXiv:2202.12837), October 2022. doi: 10.48550/arXiv.2202.12837. URL http://arxiv.org/abs/2202.12837. arXiv:2202.12837 [cs].

moire. Language models are multiverse generators, January 2021. URL https://generative.ink/posts/language-models-are-multiverse-generators/.

Murahari, V., Jimenez, C. E., Yang, R., and Narasimhan, K. Datamux: Data multiplexing for neural networks. (arXiv:2202.09318), November 2022. doi: 10.48550/arXiv.2202.09318. URL http://arxiv.org/abs/2202.09318. arXiv:2202.09318 [cs].

Nardo, C. The waluigi effect (mega-post), 2023. URL https://www.lesswrong.com/posts/D7PumeYTDPfBTp3i7/the-waluigi-effect-mega-post.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Oswald, J. V., Niklasson, E., Randazzo, E., Sacramento, J., Mordvintsev, A., Zhmoginov, A., and Vladymyrov, M. Transformers Learn In-Context by Gradient Descent. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 35151–35174. PMLR, July 2023. URL https://proceedings.mlr.press/v202/von-oswald23a.html. ISSN: 2640-3498.

Panwar, M., Ahuja, K., and Goyal, N. In-context learning through the bayesian prism, June 2023. URL https://arxiv.org/abs/2306.04891v2.

Park, J., Park, J., Xiong, Z., Lee, N., Cho, J., Oymak, S., Lee, K., and Papailiopoulos, D. Can mamba learn how to learn? a comparative study on in-context learning tasks. (arXiv:2402.04248), April 2024. doi: 10.48550/arXiv.2402.04248. URL http://arxiv.org/abs/2402.04248. arXiv:2402.04248 [cs].

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Raventos, A., Paul, M., Chen, F., and Ganguli, S. The effects of pretraining task diversity on in-context learning of ridge regression. In *ICLR Workshop on Mathematical and Empirical Understanding of Foundation Models (ME-FoMo)*, 2023.

Reddy, G. The mechanistic basis of data dependence and abrupt learning in an in-context classification task. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=aN4Jf6Cx69.

Reynolds, L. and McDonell, K. Multiversal views on language models, February 2021. URL http://arxiv.org/abs/2102.06391. arXiv:2102.06391 [cs].

Shanahan, M., McDonell, K., and Reynolds, L. Role play with large language models. *Nature*, 623(7987):493–498, 2023.

Shen, E., Fan, A., Pratt, S. M., Park, J. S., Wallingford, M., Kakade, S. M., Holtzman, A., Krishna, R., Farhadi, A., and Kusupati, A. Superposed decoding: Multiple generations from a single autoregressive inference pass. (arXiv:2405.18400), May 2024a. doi: 10.48550/arXiv.2405.18400. URL http://arxiv.org/abs/2405.18400. arXiv:2405.18400 [cs].

Shen, E., Fan, A., Pratt, S. M., Park, J. S., Wallingford, M., Kakade, S. M., Holtzman, A., Krishna, R., Farhadi, A., and Kusupati, A. Superposed decoding: Multiple generations from a single autoregressive inference pass. *arXiv preprint arXiv:2405.18400*, 2024b.

Todd, E., Li, M. L., Sharma, A. S., Mueller, A., Wallace, B. C., and Bau, D. Function vectors in large language models. (arXiv:2310.15213), February 2024. doi: 10.48550/arXiv.2310.15213. URL http://arxiv.org/abs/2310.15213. arXiv:2310.15213 [cs].

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Wei, C., Chen, Y., and Ma, T. Statistically meaningful approximation: a case study on approximating turing machines with transformers. *Advances in Neural Information Processing Systems*, 35:12071–12083, 2022.

Wei, J., Wei, J., Tay, Y., Tran, D., Webson, A., Lu, Y., Chen, X., Liu, H., Huang, D., Zhou, D., et al. Larger language models do in-context learning differently. *arXiv preprint arXiv:2303.03846*, 2023.

Wu, J., Zou, D., Chen, Z., Braverman, V., Gu, Q., and Bartlett, P. L. How many pretraining tasks are needed for in-context learning of linear regression? In *International Conference on Learning Representations (ICLR)*, 2024.

Xie, S. M., Raghunathan, A., Liang, P., and Ma, T. An explanation of in-context learning as implicit bayesian inference. (arXiv:2111.02080), July 2022a. doi: 10.48550/arXiv.2111.02080. URL http://arxiv.org/abs/2111.02080. arXiv:2111.02080 [cs].

Xie, S. M., Raghunathan, A., Liang, P., and Ma, T. An Explanation of In-context Learning as Implicit Bayesian Inference, July 2022b. URL http://arxiv.org/abs/2111.02080. arXiv:2111.02080 [cs].

Yang, L., Lee, K., Nowak, R., and Papailiopoulos, D. Looped transformers are better at learning learning algorithms. *arXiv preprint arXiv:2311.12424*, 2023.

Zeng, Y., Kang, W., Chen, Y., Koo, H. I., and Lee, K. Can mllms perform text-to-image in-context learning? (arXiv:2402.01293), April 2024. doi: 10.48550/arXiv.2402.01293. URL http://arxiv.org/abs/2402.01293. arXiv:2402.01293 [cs].

Zhang, Y., Zhang, F., Yang, Z., and Wang, Z. What and how does in-context learning learn? bayesian model averaging, parameterization, and generalization. (arXiv:2305.19420), October 2023. doi: 10.48550/arXiv.2305.19420. URL http://arxiv.org/abs/2305.19420. arXiv:2305.19420 [cs, stat].

Zhou, H., Nova, A., Larochelle, H., Courville, A., Neyshabur, B., and Sedghi, H. Teaching algorithmic reasoning via in-context learning. *arXiv preprint arXiv:2211.09066*, 2022.

Zhou, H., Bradley, A., Littwin, E., Razin, N., Saremi, O., Susskind, J., Bengio, S., and Nakkiran, P. What algorithms can transformers learn? a study in length generalization. *arXiv preprint arXiv:2310.16028*, 2023.

A. Notations

Notation	Description
K	Number of tasks
l	Length of a task's output
ℓ	layer ℓ for a model
m	Number of in-context examples
n	Length of each in-context example
$\mathcal{V}$	Token vocabulary
$\boldsymbol{g}_i(\cdot)$	Operation performed by Task i
$\boldsymbol{x}^{(j)}$	Data for example j
$\boldsymbol{y}^{(j)}$	Label for example j
$\boldsymbol{s}_m$	m in-context examples
$\boldsymbol{f}(\cdot)$	Model (predictor)
$\boldsymbol{p}$	Positional encodings

B. Task superposition capabilities as the model scales

Finding 5: *Within the same LLM family, bigger models can solve more tasks in parallel and better calibrate to ICL distribution.*

We want to further investigate how models' task superposition capabilities changes as the model size scales. In particular, we investigate two questions: 1) whether larger models can perform more tasks in-context and 2) whether larger models can align their output distribution more closely with the distribution of task examples provided in the prompt. We chose the Qwen-2.5 model family since it contains several model sizes ranging from 0.5B to 14B parameters.

We first introduce a quantity which captures the capability of a model to perform multiple tasks. Given a prompt that contains examples of K tasks, we define r to be the number of these tasks whose correct answers appear among the model's top-K most likely outputs. Note that $r \leq K$.

To see how close the model align the output distribution with the distribution of task examples, we use KL-divergence defined below:

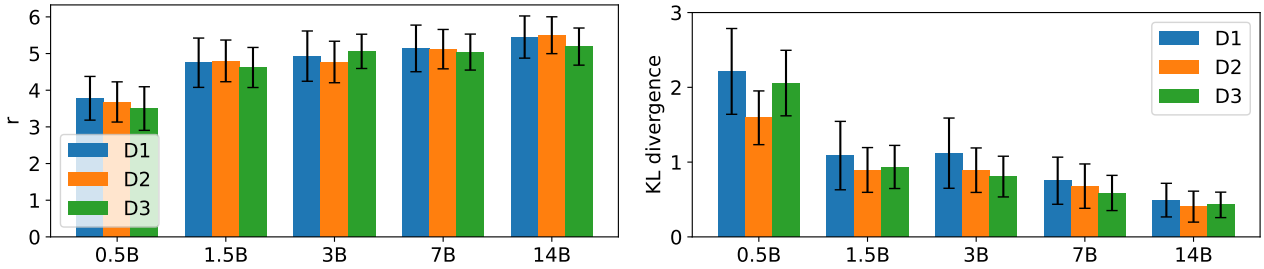
$$\mathrm{KL}(\mathcal{P}||\mathcal{D}) = \sum_{x \in X} \mathcal{P}(x) \log \left(\frac{\mathcal{P}(x)}{\mathcal{D}(x)} \right), \tag{2}$$

where $\mathcal{P}$ is the models' probabilities on the outputs when correctly performing each task on the query and $\mathcal{D}$ is the in-context task example distribution. For example the prompt in Figure 1a (left) gives $\mathcal{P} = [0.5217, 0.1316, 0.1110, 0.2169, ...]$ and $\mathcal{D} = [0.25, 0.25, 0.25, 0.25, 0, ...]$.

We consider the setting of $K = 6$ different tasks: given an input of the form "$\{\texttt{num}\}\rightarrow$" where $\texttt{num}$ is a two-digit integer, we consider 6 tasks that output (1) $\texttt{num}$ itself, (2) negation of $\texttt{num}$, (3) $\texttt{num} + 1$, (4) $\texttt{num} - 1$, (5) $\texttt{num} \times 2$ and (6) $\texttt{num}^2$.

We choose the number of in-context examples $m = 60$ (each task has 10 examples) and configure the prompt with three different in-context task example distributions $\mathcal{D}_1, \mathcal{D}_2$ and $\mathcal{D}_3$. In particular, $\mathcal{D}_1$ is the uniform distribution, $\mathcal{D}_2$ has probability 0.5 on the third task and 0.1 on other tasks, and $\mathcal{D}_3$ is a distribution with probabilities alternating between 0.25 and 0.083.

For each in-context task examples distribution $\mathcal{D}_i$, we generate 100 prompts and for each prompt we calculate the probabilities of outputs when correctly performing each task. The average values of r and KL-divergence under three distributions are shown in Figure 6.



(a) r (the number of tasks whose correct answers appear in top-K most likely outputs).

(b) KL divergence.

Figure 6: (a) Average number of tasks completed, r, and (b) KL divergence for Qwen-2.5 model family under ICL distributions $\mathcal{D}_1, \mathcal{D}_2$ and $\mathcal{D}_3$ where $\mathcal{D}_1$ is the uniform distribution, $\mathcal{D}_2$ has probability 0.5 on the third task and 0.1 on other tasks, and $\mathcal{D}_3$ is a distribution with probabilities alternating between 0.25 and 0.083.

In Figure 6a, we can observe that in general bigger models have higher r values. This shows bigger models will have more correct answers of tasks show up in their top-K probable outputs and therefore they can solve more tasks at the same time. In Figure 6b, we can see that for larger models like Qwen-2.5 7B and Qwen-2.5 14B, the KL-divergence values are small, and for each model, the differences between KL-divergence values under in-context task example distributions $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$ are small. This indicates that bigger models can better calibrate their output distribution to the in-context task example distribution.

C. Implementation details on calculating probabilities

In this section we provide details on how we calculate probabilities of different outputs given a prompt in our setting.

Notations. Let $\mathcal{V}$ be the token vocabulary, `LM` be an LLM, T be the tokenizer. We use "..." to represent a string, `<...>` to represent a single token where the content within the angle brackets is an integer representing token's index in vocabulary. For example, token `<266>` corresponds to "at". We use [`<...>, ..., <...>`] to represent a sequence of tokens. Given a tokenizer, we use two functions $\texttt{tok}(\cdot)$ and $\texttt{detok}(\cdot)$ to tokenize strings and detokenize tokens. For example $\texttt{tok}$("superposition") = [`<9712>`,`<3571>`] and $\texttt{detok}$([`<16>`,`<10>`,`<16>`,`<28>`,`<17>`]) = "1+1=2".

In our in-context learning setting, an input string consists of in-context examples (separated by the delimiter "\n") and a query. For example, an example prompt can be "1+1=2\n2+2=4\n3+3=".

We view an LLM as a next-token predictor that outputs a probability distribution over the token space given input and there is a corresponding $\mathbb{P}(\cdot|\cdot)$ such that given a sequence of tokens $[v_1, ..., v_M]$ where $v_j \in \mathcal{V}$, $\mathbb{P}(u \mid [v_1, ..., v_M])$ measures the probability of the next token being u where $u \in \mathcal{V}$.

Measuring the probabilities of task answers. Let I be the input prompt. For example, in the example in Figure 1a (left), the prompt is "11+26->37\n33+13->quarante-six\n ...30+25->fifty-five\n91+83->". We consider four tasks: 1) numerical addition, 2) addition in English, 3) addition in French and 4) addition in Spanish. The corresponding task answers (the output of correctly performing task on the query) are "174", "one hundred and seventy-four", "cent soixante-quatorze" and "ciento setenta y cuatro", respectively. We want to measure the probability of each task answer.

Let o be a task answer in string. Let $[v_1, ..., v_M] := \texttt{tok}(I)$ and let $[u_1, ..., u_N] := \texttt{tok}(o)$. Then the probability of the task answer o given prompt I can be calculated as

$$\mathbb{P}(u_1 \mid [v_1, ..., v_M]) \prod_{j=2}^{N} \mathbb{P}(u_j \mid [v_1, ..., v_M, u_1, ..., u_{j-1}]). \tag{3}$$

D. Implementation details on task vectors

We use the task vector definition from Hendel et al. (2023). For example, for task `copy(op1)` in Figure 1b (left), the procedure to collect the task vector consists of

1. Collect a dataset of 100 ICL sample prompts. Each prompt consists of $m = 60$ in-context examples of a particular task and a query $\boldsymbol{x}^{(m+1)}$. Each task example $(\boldsymbol{x}^{(j)}, \boldsymbol{y}^{(j)})$ follows the form "`{op1}@{op2}={op1}`", where $\boldsymbol{x}^{(j)}$ has the form "`{op1}@{op2}=`" and $\boldsymbol{y}^{(j)}$ is performing task `copy(op1)` on $\boldsymbol{x}^{(j)}$, namely `op1`.
2. For each prompt $\equiv \boldsymbol{s} = [\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}, ..., \boldsymbol{x}^{(m)}, \boldsymbol{y}^{(m)}, \boldsymbol{x}^{(m+1)}]$ in the dataset, we feed $\boldsymbol{s}$ into the transformer model $\boldsymbol{f}$, and extract the feature (which is a vector) at the last "=" token in layer ℓ. Call this vector $\boldsymbol{f}(\boldsymbol{s}; \ell)$. Then we average $\boldsymbol{f}(\boldsymbol{s}; \ell)$ across all prompt $\boldsymbol{s}$ to get $\boldsymbol{v}(\ell)$ for layer ℓ.
3. Now for each layer ℓ we have a vector $\boldsymbol{v}(\ell)$. We run a forward pass with one query $\boldsymbol{x}$ in the form "`{op1}@{op2}=`" and we patch in $\boldsymbol{v}(\ell)$ at the "=" token position in layer ℓ, simulating the effect of a complete context. We repeat this process 100 times for different query $\boldsymbol{x}$ and get an accuracy acc_ℓ of performing task `copy(op1)` with vector $\boldsymbol{v}(\ell)$.
4. The task vector layer ℓ^* is selected by
$$\ell^* = \arg\max_\ell acc_\ell,$$
and we define the task vector $\mathrm{V}_{\texttt{copy(op1)}} := \boldsymbol{v} = \boldsymbol{v}(\ell^*)$.

Here we record the task vector layer where task vectors are extracted in Section 7.

Task	Task vector layer
`copy(op1)`, `copy(op2)`, `op1+op2`	14
`to_de(op1)`, `to_fr(op1)`, `to_it(op1)`	19

Table 1: Task vector layer for various tasks considered in Section 7.

E. Construction displaying superposition

In this section we construct a Transformer that is performing superposition of multiple tasks at inference. For this purpose, we first construct a Transformer that copies from n-tuple in-context examples the i-th one, as well as any function using the ReLU layers. We then create indicator vectors, for each task, which show whether a specific task is present in-context or not. As a last step, we combine these indicator vectors to create the superposition of different tasks. Notice that using the parallel heads of the transformer architecture we can process each task independently until the last step in which the predictions are combined.

E.1. Overview

Here we provide a brief overview of how the construction is implemented, while latter we provide the corresponding details.

Prediction based on multiple tasks. Assume that we are given m in-context samples $(\boldsymbol{x}_1^{(j)}, \dots, \boldsymbol{x}_{n-2}^{(j)}, \text{'='}, \boldsymbol{y}^{(j)})_{j=1}^m$ where '=' represents a specific value used only for preceding the label, and a set of k different Transformers TF_i which can implement the T different desired tasks, where each deterministic task is denoted as $\boldsymbol{g}_i(\boldsymbol{x}^{(j)})$ with $i \in [k]$ and $j \in [m]$, *i.e.* $\boldsymbol{y}^{(j)} = \boldsymbol{g}_i(\boldsymbol{x}^{(j)})$ for some task i dependent on sample j. Using the weights of each TF_i, we can compute outputs of the following form:

$$\begin{bmatrix} \dots & \boldsymbol{x}_1^{(j)} & \dots & \boldsymbol{x}_{n-2}^{(j)} & \textbf{=} & \boldsymbol{y}^{(j)} & \dots \\ \dots & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots \\ & \vdots & & \vdots & \vdots & \vdots & \\ \dots & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots \end{bmatrix} \rightarrow \begin{bmatrix} \dots & \boldsymbol{x}_1^{(j)} & \dots & \boldsymbol{x}_{n-2}^{(j)} & \textbf{=} & \boldsymbol{y}^{(j)} & \dots \\ \dots & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \|\boldsymbol{g}_1(\boldsymbol{x}^{(j)}) - \boldsymbol{y}^{(j)}\|_1 & \dots \\ & \vdots & & \vdots & \vdots & \vdots & \\ \dots & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \|\boldsymbol{g}_T(\boldsymbol{x}^{(j)}) - \boldsymbol{y}^{(j)}\|_1 & \dots \end{bmatrix}$$

We use the l_1 norm to aggregate the prediction, in case that the task is multi-dimensional. These differences are used to identify tasks, as $\|\boldsymbol{g}_i(\boldsymbol{x}^{(j)}) - \boldsymbol{y}^{(j)}\|_1 \approx 0$ for $\boldsymbol{y}^{(j)}$ coming from task i. Different heads at each layer in the model are used to execute each of the tasks in parallel using the weights from TF_i. In Appendix E we construct tasks where an arbitrary function $\boldsymbol{g}_i(\boldsymbol{x}_l^{(j)})$ is implemented using ReLUs for some fixed l that is task-specific.

Creating task identifiers. Having the differences between the implemented function and the label, we first use the ReLUs to clean up the vectors $\boldsymbol{v}_k$ so that only the positions in each vector that are associated with a task are maintained and the rest are set to 1[5]. We thus create the vectors $(\boldsymbol{v}'_k)_i = \|\boldsymbol{g}_k(\boldsymbol{x}_{1:l-1}^{(j)}) - \boldsymbol{x}_l^{(j)}\|_1$ and $(\boldsymbol{v}'_*)_* = 1$ otherwise. Now we use ReLUs to threshold and create an indicator vectors $\mathbf{1}_{\{\|\boldsymbol{g}_k(\boldsymbol{x}_{1:l-1}^{(j)}) - \boldsymbol{x}_l^{(j)}\|_1 \approx 0\}}$ which identify the task, *i.e.* these are task identifiers. Notice that if the task is correctly predicted then the difference should be close to 0 (up to some error), while if the task is not identified the corresponding value would not be 0; the rest of the rows would be 1. We have created one vector for each task, which has 1 in the position of the corresponding task if the task was identified in the context.

Averaging and task superposition. As a last step, we average all the task identifiers and place the result in the last column, in which the next prediction will happen. We then use the averaged task identifier to weight the prediction of each task based on it, as in task superposition. If the task has been identified multiple times in the context, it would be assigned a higher weight/probability.

E.2. Task Identification

The first task for performing task superposition based on in-context examples is to define a set of tasks that the model is able to implement.

First, the outputs of tasks need to be identified.

Lemma 1. *Consider the following input*

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^{(1)} & \dots & \boldsymbol{y}^{(j-1)} & \boldsymbol{x}_1^{(j)} & \dots & \boldsymbol{x}_{n-2}^{(j)} & \textbf{=} & \boldsymbol{y}^{(j)} & \boldsymbol{x}_1^{(j+1)} & \dots \\ 0 & \dots & 0 & 0 & \dots & 0 & 1 & 0 & 0 & \dots \\ \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots \end{bmatrix},$$

[5]This step is not mandatory, but it ensures that we have no values over which we have no control. We leave as future work an error analysis on how these values could affect the task identifiers

where $\boldsymbol{x}_i^{(j)} \in \mathbb{R}^{d-1}$ before the positional encodings are added, with one additional dimension that represents if the symbol is an 'equals' symbol. Then, a 1-layer transformer with a single attention head and embedding dimension $\mathcal{O}(d + \log(mn))$ can output

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^{(1)} & \dots & \boldsymbol{y}^{(j-1)} & \boldsymbol{x}_1^{(j)} & \dots & \boldsymbol{x}_{n-2}^{(j)} & \mathbf{=} & \boldsymbol{y}^{(j)} & \boldsymbol{x}_1^{(j+1)} & \dots \\ 0 & \dots & 1 & 0 & \dots & 0 & 0 & 1 & 0 & \dots \end{bmatrix}$$

Proof. With positional encodings appended, let the input have the following structure:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^{(1)} & \dots & \boldsymbol{x}_1^{(j)} & \boldsymbol{x}_2^{(j)} & \dots & \boldsymbol{x}_{n-2}^{(j)} & \mathbf{=} & \boldsymbol{y}^{(j)} & \boldsymbol{x}_1^{(j+1)} & \dots \\ 0 & \dots & 0 & 0 & \dots & 0 & 1 & 0 & 0 & \dots \\ \boldsymbol{p}_{n+1} & \cdots & \boldsymbol{p}_{jn+1} & \boldsymbol{p}_{jn+2} & \cdots & \boldsymbol{p}_{jn+n-2} & \boldsymbol{p}_{jn+n-1} & \boldsymbol{p}_{jn+n} & \boldsymbol{p}_{(j+1)n+1} & \cdots \\ \boldsymbol{p}_n & \cdots & \boldsymbol{p}_{jn} & \boldsymbol{p}_{jn+1} & \cdots & \boldsymbol{p}_{jn+n-3} & \boldsymbol{p}_{jn+n-2} & \boldsymbol{p}_{jn+n-1} & \boldsymbol{p}_{(j+1)n} & \cdots \end{bmatrix} \tag{4}$$

To rotate the second row one position to the right, use the following matrices.

$$\boldsymbol{W}_Q = \begin{bmatrix} \mathbf{0} & 0 & \mathbf{0} & \mathbb{I} \end{bmatrix}$$
$$\boldsymbol{W}_K = \begin{bmatrix} \mathbf{0} & 0 & C\mathbb{I} & \mathbf{0} \end{bmatrix}$$
$$\boldsymbol{W}_V = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

The pair $\boldsymbol{W}_Q$ and $\boldsymbol{W}_K$ attend tokens to the token directly to the right. The value matrix simply filters only the second row in-place. A second head can used to clear the original 1s, resulting in

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^{(1)} & \dots & \boldsymbol{x}_1^{(j)} & \boldsymbol{x}_2^{(j)} & \dots & \boldsymbol{x}_{n-2}^{(j)} & \mathbf{=} & \boldsymbol{y}^{(j)} & \boldsymbol{x}_1^{(j+1)} & \dots \\ 0 & \dots & 0 & 0 & \dots & 0 & 0 & 1 & 0 & \dots \\ \boldsymbol{p}_{n+1} & \cdots & \boldsymbol{p}_{jn+1} & \boldsymbol{p}_{jn+2} & \cdots & \boldsymbol{p}_{jn+n-2} & \boldsymbol{p}_{jn+n-1} & \boldsymbol{p}_{jn+n} & \boldsymbol{p}_{(j+1)n+1} & \cdots \\ \boldsymbol{p}_n & \cdots & \boldsymbol{p}_{jn} & \boldsymbol{p}_{jn+1} & \cdots & \boldsymbol{p}_{jn+n-3} & \boldsymbol{p}_{jn+n-2} & \boldsymbol{p}_{jn+n-1} & \boldsymbol{p}_{(j+1)n} & \cdots \end{bmatrix}, \tag{5}$$

as desired. □

Implementation of functions. To illustrate a set of operations that could be implemented with a transformer, we consider approximating functions as sums of ReLUs; we use a result from Bai et al. (2023b), which we present below.

Definition 1 (Definition 12 in (Bai et al., 2023b))**.** *A function $g : \mathbb{R}^k \to \mathbb{R}$ is (ϵ, R, M, C)-approximable by sum of ReLUs, if there exists an "(M,C)-sum of ReLUs" function*

$$f_{M,C}(\boldsymbol{x}) = \sum_{m=1}^{M} c_m \mathit{ReLU}(\boldsymbol{a}_m^\top [\boldsymbol{x}; 1]) \text{ with } \sum_{m=1}^{M} |c_m| \le C, \ \max_{m \in [M]} \|\boldsymbol{a}_m\|_1 \le 1, \ \boldsymbol{a}_m \in \mathbb{R}^{k+1}, \ c_m \in \mathbb{R}$$

such that $\sup_{\boldsymbol{x} \in [-R,R]^k} \left| g(\boldsymbol{x}) - f_{(M,C)}(\boldsymbol{x}) \right| \le \epsilon$.

Definition 2 (Definition A.1 in (Bai et al., 2023b))**.** *We say a function $g : \mathbb{R}^k \to \mathbb{R}$ is (R, C_l)-smooth if for $s = \lceil (k-1)/2 \rceil + 2$, $g \in C^2$[6] on $[-R, R]^k$ and*

$$\sup_{\boldsymbol{x} \in [-R,R]^k} \left\| \nabla^i g(\boldsymbol{x}) \right\|_\infty = \sup_{\boldsymbol{x} \in [-R,R]^k} \max_{j1,\dots,j_i \in [k]} \left| \partial_{x_{j1},\dots,x_{ji}} g(\boldsymbol{x}) \right| \le L_i$$

for all $i = 0, 1, 2$ with $\max_{0 \le i \le s} L_i R^i \le C_l$.

Proposition 1 (Proposition A.1 in (Bai et al., 2023b))**.** *For any $\epsilon > 0$, $R \ge 1$, $C_l > 0$, we have that: Any (R, C_l)-smooth function, $g : \mathbb{R} \to \mathbb{R}$ is (ϵ, R, M, C)-approximable by sum of ReLUs (Definition 1) with $M \le C(k) C_l^2 \log(1 + C_l \epsilon)/\epsilon^2$.*

[6] C^i denotes that a function is i times differentiable with continuous i-th derivative.

Lemma 2. *For any function $g : \mathbb{R}^k \to \mathbb{R}$ that is (R, C_l)-smooth, there exists a transformer with two layers, one head and width $\mathcal{O}(\log(n) + d)$, where d satisfies the requirements of Prop. 1, such that given as input*

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^{(1)} & \dots & \boldsymbol{y}^{(j-1)} & \boldsymbol{x}_1^{(j)} & \dots & \boldsymbol{x}_{n-2}^{(j)} & \mathbf{=} & \boldsymbol{y}^{(j)} & \boldsymbol{x}_1^{(j+1)} & \dots \\ 0 & \dots & 1 & 0 & \dots & 0 & 0 & 1 & 0 & \dots \\ \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots \end{bmatrix},$$

it outputs

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^{(1)} & \dots & \boldsymbol{y}^{(j-1)} & \boldsymbol{x}_1^{(j)} & \dots & \boldsymbol{x}_{n-2}^{(j)} & \mathbf{=} & \boldsymbol{y}^{(j)} & \boldsymbol{x}_1^{(j+1)} & \dots \\ * & \dots & * & * & \dots & * & * & * & * & \dots \\ \mathbf{0} & \dots & \tilde{g}(\boldsymbol{x}_i^{(j-1)}) - \boldsymbol{y}^{(j-1)} & * & \dots & * & * & \tilde{g}(\boldsymbol{x}_i^{(j)}) - \boldsymbol{y}^{(j)} & * & \dots \end{bmatrix}$$

where $|\tilde{g}(x) - g(x)| \leq \epsilon$ and for some $i \in [1, \dots, n-2]$.

Proof. We consider that the positional encodings are added in the input and we have

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^{(1)} & \dots & \boldsymbol{x}_1^{(j)} & \boldsymbol{x}_2^{(j)} & \dots & \boldsymbol{x}_{n-2}^{(j)} & \mathbf{=} & \boldsymbol{y}^{(j)} & \boldsymbol{x}_1^{(j+1)} & \dots \\ 0 & \dots & 0 & 0 & \dots & 0 & 0 & 1 & 0 & \dots \\ \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots \\ 1 & \dots & 1 & 1 & \dots & 1 & 1 & 1 & 1 & \dots \\ \boldsymbol{p}_{n+1} & \cdots & \boldsymbol{p}_{jn+1} & \boldsymbol{p}_{jn+2} & \cdots & \boldsymbol{p}_{jn+n-2} & \boldsymbol{p}_{jn+n-1} & \boldsymbol{p}_{jn+n} & \boldsymbol{p}_{(j+1)n+1} & \cdots \\ \boldsymbol{p}_{n+1-s} & \cdots & \boldsymbol{p}_{jn+1-s} & \boldsymbol{p}_{jn+2-s} & \cdots & \boldsymbol{p}_{jn+n-2-s} & \boldsymbol{p}_{jn+n-1-s} & \boldsymbol{p}_{jn+n-s} & \boldsymbol{p}_{(j+1)n+1-s} & \cdots \end{bmatrix} \tag{6}$$

where we fix some positional encodings $\boldsymbol{p}_k$ where $\boldsymbol{p}_k^\top \boldsymbol{p}_k$ is larger than $\boldsymbol{p}_k^\top \boldsymbol{p}_l$ by some threshold for $k \neq l$. The encodings used here are the binary representations of $k \in \{-1, 1\}^{\log(mn)}$. Further, we consider 1s in the positions with the results of the task to differentiate the context of the task and the result of the task. Define $s = n - i$, the distance between the result and the associated value in the context.

In the first layer, we use the MLP's to create $\tilde{g}$ according to Proposition 1

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^{(1)} & \dots & \boldsymbol{x}_1^{(j)} & \boldsymbol{x}_2^{(j)} & \dots & \boldsymbol{x}_{n-2}^{(j)} & \mathbf{=} & \boldsymbol{y}^{(j)} & \boldsymbol{x}_1^{(j+1)} & \dots \\ 0 & \dots & 0 & 0 & \dots & 0 & 0 & 1 & 0 & \dots \\ \tilde{g}(\boldsymbol{x}_1^{(1)}) & \dots & \tilde{g}(\boldsymbol{x}_1^{(j)}) & \tilde{g}(\boldsymbol{x}_2^{(j)}) & \dots & \tilde{g}(\boldsymbol{x}_{n-2}^{(j)}) & \tilde{g}(\mathbf{=}) & \tilde{g}(\boldsymbol{y}^{(j)}) & \tilde{g}(\boldsymbol{x}_1^{(j+1)}) & \dots \\ \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots \\ 1 & \dots & 1 & 1 & \dots & 1 & 1 & 1 & 1 & \dots \\ \boldsymbol{p}_{n+1} & \cdots & \boldsymbol{p}_{jn+1} & \boldsymbol{p}_{jn+2} & \cdots & \boldsymbol{p}_{jn+n-2} & \boldsymbol{p}_{jn+n-1} & \boldsymbol{p}_{jn+n} & \boldsymbol{p}_{(j+1)n+1} & \cdots \\ \boldsymbol{p}_{n+1-s} & \cdots & \boldsymbol{p}_{jn+1-s} & \boldsymbol{p}_{jn+2-s} & \cdots & \boldsymbol{p}_{jn+n-2-s} & \boldsymbol{p}_{jn+n-1-s} & \boldsymbol{p}_{jn+n-s} & \boldsymbol{p}_{(j+1)n+1-s} & \cdots \end{bmatrix} \tag{7}$$

The next operation is a shift of the sequence of $\tilde{g}(\cdot)$'s to the right by s. This will align the desired output $\tilde{g}(\boldsymbol{x}_i^{(j)})$ with the observed output $\boldsymbol{y}^{(j)}$. Consider the following weight matrices

$$\boldsymbol{W}_Q = \begin{bmatrix} \dots & 0 & \mathbf{0} & \mathbb{I} \end{bmatrix} \tag{8}$$

$$\boldsymbol{W}_K = \begin{bmatrix} \dots & 0 & C\mathbb{I} & \mathbf{0} \end{bmatrix} \tag{9}$$

$$\boldsymbol{W}_V = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbb{I} & \dots & \mathbf{0} \\ \vdots & \vdots & \vdots & & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \end{bmatrix} \tag{10}$$

for some large constant C to decrease error from the softmax attending to the incorrect tokens. This produces (within a small error induced by using a softmax)

$$(\boldsymbol{X}^\top \boldsymbol{W}_K^\top \boldsymbol{W}_Q \boldsymbol{X})_{i,j} = \boldsymbol{p}_{n+i}^\top \boldsymbol{p}_{n-s+j} \tag{11}$$

$$\sigma_S(\boldsymbol{X}^\top \boldsymbol{W}_K^\top \boldsymbol{W}_Q X)_{i,j} = \mathbb{1}_{\{n+i=n-s+j\}} = \mathbb{1}_{\{i=j-s\}} \tag{12}$$

$$\boldsymbol{W}_V \boldsymbol{X} = \begin{bmatrix} \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots \\ \dots & 0 & 0 & \dots & 0 & 0 & 0 & \dots \\ \dots & \tilde{g}(\boldsymbol{x}_1^{(j)}) & \tilde{g}(\boldsymbol{x}_2^{(j)}) & \dots & \tilde{g}(\boldsymbol{x}_{n-2}^{(j)}) & \tilde{g}(\mathbf{=}) & \tilde{g}(\boldsymbol{y}^{(j)}) & \dots \\ & \vdots & \vdots & & \vdots & \vdots & \vdots & \\ \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots \end{bmatrix} \tag{13}$$

$$\boldsymbol{W}_V \boldsymbol{X} \sigma_S(\boldsymbol{X}^\top \boldsymbol{W}_K^\top \boldsymbol{W}_Q \boldsymbol{X}) = \begin{bmatrix} \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots \\ \dots & 0 & 0 & \dots & 0 & 0 & 0 & \dots \\ \dots & * & * & \dots & * & * & \tilde{g}(\boldsymbol{x}_i^{(j)}) & \dots \\ & \vdots & \vdots & & \vdots & \vdots & \vdots & \\ \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots \end{bmatrix} \tag{14}$$

$$\boldsymbol{X} + \boldsymbol{W}_V \boldsymbol{X} \sigma_S(\boldsymbol{X}^\top \boldsymbol{W}_K^\top \boldsymbol{W}_Q \boldsymbol{X}) = \begin{bmatrix} \dots & \boldsymbol{x}_1^{(j)} & \boldsymbol{x}_2^{(j)} & \dots & \boldsymbol{x}_{n-1}^{(j)} & \mathbf{=} & \boldsymbol{y}^{(j)} & \dots \\ \dots & 0 & 0 & \dots & 0 & 0 & 1 & \dots \\ \dots & * & * & \dots & * & * & \tilde{g}(\boldsymbol{x}_i^{(j)}) & \dots \\ \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots \\ \dots & 1 & 1 & \dots & 1 & 1 & 1 & \dots \\ \dots & \boldsymbol{p}_{jn+1} & \boldsymbol{p}_{jn+2} & \dots & \boldsymbol{p}_{jn+n-2} & \boldsymbol{p}_{jn+n-1} & \boldsymbol{p}_{jn+n} & \dots \\ \dots & \boldsymbol{p}_{jn+1-s} & \boldsymbol{p}_{jn+2-s} & \dots & \boldsymbol{p}_{jn+n-2-s} & \boldsymbol{p}_{jn+n-1-s} & \boldsymbol{p}_{jn+n-s} & \dots \end{bmatrix} \tag{15}$$

$$\tag{16}$$

Each matrix above only shows the slice that contains the j-th in-context example. This is repeated for each of the other in-context examples.

As a final step with an MLP, subtract row 1 from row 3 to achieve the following output:

$$\begin{bmatrix} \dots & \boldsymbol{x}_1^{(j)} & \boldsymbol{x}_2^{(j)} & \dots & \boldsymbol{x}_{n-1}^{(j)} & \mathbf{=} & \boldsymbol{y}^{(j)} & \dots \\ \dots & 0 & 0 & \dots & 0 & 0 & 1 & \dots \\ \dots & * & * & \dots & * & * & \tilde{g}(\boldsymbol{x}_i^{(j)}) - \boldsymbol{y}^{(j)} & \dots \\ \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots \\ \dots & 1 & 1 & \dots & 1 & 1 & 1 & \dots \\ \dots & \boldsymbol{p}_{jn+1} & \boldsymbol{p}_{jn+2} & \dots & \boldsymbol{p}_{jn+n-2} & \boldsymbol{p}_{jn+n-1} & \boldsymbol{p}_{jn+n} & \dots \\ \dots & \boldsymbol{p}_{jn+1-s} & \boldsymbol{p}_{jn+2-s} & \dots & \boldsymbol{p}_{jn+n-2-s} & \boldsymbol{p}_{jn+n-1-s} & \boldsymbol{p}_{jn+n-s} & \dots \end{bmatrix} \tag{17}$$

□

Copy Tasks As has been experimentally investigated, the situation where a specific position within the context is copied as the label can be easily implemented by setting $g(\boldsymbol{x}) = \boldsymbol{x}$. The dependence on the subscript i within the construction is what allows the position copied to vary.

E.2.1. Identifying if Task's Output Matches the In-context Example

Lemma 3. *A three layer transformer with ReLU MLPs and embedding dimension $\mathcal{O}(d + \log(mn))$ can calculate the proportion of in context examples that come from a specific task, where m is the number of in-context examples, each of length n and dimension d.*

Proof. We now have a matrix of the following form.

$$\begin{bmatrix} \dots & \boldsymbol{x}_1^{(j)} & \boldsymbol{x}_2^{(j)} & \dots & = & \boldsymbol{y}^{(j)} & \dots \\ \dots & 0 & 0 & \dots & 0 & 1 & \dots \\ \dots & * & * & \dots & * & f(\boldsymbol{x}_\cdot^{(j)}) - \boldsymbol{y}^{(j)} & \dots \\ \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots \\ \dots & 1 & 1 & \dots & 1 & 1 & \dots \\ \dots & \boldsymbol{p}_{jn+1} & \boldsymbol{p}_{jn+2} & \dots & \boldsymbol{p}_{jn+n-1} & \boldsymbol{p}_{jn+n} & \dots \\ \dots & \boldsymbol{p}_{jn+1-s} & \boldsymbol{p}_{jn+2-s} & \dots & \boldsymbol{p}_{jn+n-1-s} & \boldsymbol{p}_{jn+n-s} & \dots \end{bmatrix} \tag{18}$$

If the task is correct, than $f(\boldsymbol{x}_\cdot^{(j)}) - \boldsymbol{y}^{(j)} \approx \mathbf{0}$, with some small error coming from softmaxs and function approximation error. First, we find the $L1$-norm of $f(\boldsymbol{x}_\cdot^{(j)}) - \boldsymbol{y}^{(j)}$ using an MLP. For calculating $\|\boldsymbol{z}\|_1$ for arbitrary $\boldsymbol{z}$, we can use

$$\|\boldsymbol{z}\|_1 = \sum_{i=1}^{d} \text{ReLU}(\boldsymbol{z}_i) - \text{ReLU}(-\boldsymbol{z}_i) \tag{19}$$

which can be done in a single 1-layer MLP. Thus, we have

$$\begin{bmatrix} \dots & \boldsymbol{x}_1^{(j)} & \boldsymbol{x}_2^{(j)} & \dots & = & \boldsymbol{y}^{(j)} & \dots \\ \dots & 0 & 0 & \dots & 0 & 1 & \dots \\ \dots & * & * & \dots & * & f(\boldsymbol{x}_\cdot^{(j)}) - \boldsymbol{y}^{(j)} & \dots \\ \dots & * & * & \dots & * & \|f(\boldsymbol{x}_\cdot^{(j)}) - \boldsymbol{y}^{(j)}\|_1 & \dots \\ \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots \\ \dots & 1 & 1 & \dots & 1 & 1 & \dots \\ \dots & \boldsymbol{p}_{jn+1} & \boldsymbol{p}_{jn+2} & \dots & \boldsymbol{p}_{jn+n-1} & \boldsymbol{p}_{jn+n} & \dots \\ \dots & \boldsymbol{p}_{jn+1-s} & \boldsymbol{p}_{jn+2-s} & \dots & \boldsymbol{p}_{jn+n-1-s} & \boldsymbol{p}_{jn+n-s} & \dots \end{bmatrix} \tag{20}$$

Notice that if some task has different dimension than another task, the "extra" rows would be zero and will not affect the result.

For clarity, we set all $*$ values in the $\|\cdot\|_1$ row to 1s. These will cause the following $\hat{\delta}$ in the following set these to 0. This operation can be omitted as the construction handles these trash values at a later layer.

Let b represent the value of the flag in the second row marking the $\boldsymbol{y}$ vectors and let x represent the values in the row with $\|f(\boldsymbol{x}^{(j)}) - \boldsymbol{y}^{(j)}\|_1$. The following ReLUs set the * values to 1.

$$x \leftarrow x + 1 - \text{ReLU}(x - Cb) - \text{ReLU}(Cb - C + 1) \tag{21}$$

for some large constant C. When $b = 0$, this reduces to $x + 1 - x = 1$, and when $b = 1$, this reduces to $x + 1 - 1 = x$, as desired.

$$\begin{bmatrix} \dots & \boldsymbol{x}_1^{(j)} & \boldsymbol{x}_2^{(j)} & \dots & = & \boldsymbol{y}^{(j)} & \dots \\ \dots & 0 & 0 & \dots & 0 & 1 & \dots \\ \dots & * & * & \dots & * & f(\boldsymbol{x}_\cdot^{(j)}) - \boldsymbol{y}^{(j)} & \dots \\ \dots & 1 & 1 & \dots & 1 & \|f(\boldsymbol{x}_\cdot^{(j)}) - \boldsymbol{y}^{(j)}\|_1 & \dots \\ \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots \\ \dots & 1 & 1 & \dots & 1 & 1 & \dots \\ \dots & \boldsymbol{p}_{jn+1} & \boldsymbol{p}_{jn+2} & \dots & \boldsymbol{p}_{jn+n-1} & \boldsymbol{p}_{jn+n} & \dots \\ \dots & \boldsymbol{p}_{jn+1-s} & \boldsymbol{p}_{jn+2-s} & \dots & \boldsymbol{p}_{jn+n-1-s} & \boldsymbol{p}_{jn+n-s} & \dots \end{bmatrix} \tag{22}$$

Now define a thresholding function $\hat{\delta}(z)$ that satisfies $\hat{\delta}(0) = 1$ and $\hat{\delta}(z) = 0$ for $z >> 0$. One such function used here is

$$\hat{\delta}_C(z) = \text{ReLU}(1 - Cz) \tag{23}$$

for some constant C, where larger C captures a narrower neighborhood of 0.

However, a slight change needs to be added to $\hat{\delta}_C$. In the same row as $\|f(\boldsymbol{x}^{(j)}) - \boldsymbol{y}^{(j)}\|$ are many values that need to be discarded. Let b be the bit for the current column marking if the column contains an $\boldsymbol{x}$ or a $\boldsymbol{y}$. We use instead

$$\hat{\delta}_C(b, z) = \text{ReLU}(b - Cz) \tag{24}$$

This will be zero whenever $b = 0$ and $z \geq 0$. We then have as output

$$\begin{bmatrix} \dots & \boldsymbol{x}_1^{(j)} & \boldsymbol{x}_2^{(j)} & \dots & \mathbf{=} & \boldsymbol{y}^{(j)} & \dots \\ \dots & 0 & 0 & \dots & 0 & 1 & \dots \\ \dots & * & * & \dots & * & f(\boldsymbol{x}_\cdot^{(j)}) - \boldsymbol{y}^{(j)} & \dots \\ \dots & 1 & 1 & \dots & 1 & \|f(\boldsymbol{x}_\cdot^{(j)}) - \boldsymbol{y}^{(j)}\|_1 & \dots \\ \dots & 0 & 0 & \dots & 0 & \hat{\delta}_C(\|f(\boldsymbol{x}_\cdot^{(j)}) - \boldsymbol{y}^{(j)}\|_1) & \dots \\ \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots \\ \dots & 1 & 1 & \dots & 1 & 1 & \dots \\ \dots & \boldsymbol{p}_{jn+1} & \boldsymbol{p}_{jn+2} & \dots & \boldsymbol{p}_{jn+n-1} & \boldsymbol{p}_{jn+n} & \dots \\ \dots & \boldsymbol{p}_{jn+1-s} & \boldsymbol{p}_{jn+2-s} & \dots & \boldsymbol{p}_{jn+n-1-s} & \boldsymbol{p}_{jn+n-s} & \dots \end{bmatrix} \tag{25}$$

Importantly, $\hat{\delta}_C(\|f(\boldsymbol{x}_\cdot^{(j)}) - \boldsymbol{y}^{(j)}\|_1) = 1$ when $f(\cdot)$ is the correct task and $\hat{\delta}_C(\|f(\boldsymbol{x}_\cdot^{(j)}) - \boldsymbol{y}^{(j)}\|_1) = 0$ when $f(\cdot)$ disagrees by more than $\frac{1}{C}$ in $L1$-norm.

Lastly, for the next step in the construction, we need to average these soft indicators $\hat{\delta}$ to see how common f is within the context. This is done with an attention layer. Let $\boldsymbol{W}_Q$ select the row with all 1s multiplied by some large constant C, and let $\boldsymbol{W}_K$ select the row with flags for results $\boldsymbol{y}$. Then

$$\boldsymbol{X}^\top \boldsymbol{W}_K^\top \boldsymbol{W}_Q \boldsymbol{X} = \begin{bmatrix} \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 \\ C & C & \dots & C & C \\ \vdots & \vdots & & \vdots & \vdots \end{bmatrix} \tag{26}$$

$$\sigma_S(\boldsymbol{X}^\top \boldsymbol{W}_K^\top \boldsymbol{W}_Q \boldsymbol{X}) \approx \begin{bmatrix} \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 \\ 1/m & 1/m & \dots & 1/m & 1/m \\ \vdots & \vdots & & \vdots & \vdots \end{bmatrix} \tag{27}$$

where a $1/m$ will appear in every row corresponding to a result $\boldsymbol{y}$. Let the value matrix select the row containing $\hat{\delta}(\|f(\boldsymbol{x}^{(j)} - \boldsymbol{y}^{(j)})\|_1)$. Denote $p = \frac{1}{m}\sum_{j=1}^{m} \hat{\delta}(\|f(\boldsymbol{x}^{(j)} - \boldsymbol{y}^{(j)})\|_1)$. Without causal masking, we would have as output

$$\begin{bmatrix} \dots & \boldsymbol{x}_1^{(j)} & \boldsymbol{x}_2^{(j)} & \dots & \mathbf{=} & \boldsymbol{y}^{(j)} & \dots \\ \dots & 0 & 0 & \dots & 0 & 1 & \dots \\ \dots & * & * & \dots & * & f(\boldsymbol{x}_\cdot^{(j)}) - \boldsymbol{y}^{(j)} & \dots \\ \dots & 1 & 1 & \dots & 1 & \|f(\boldsymbol{x}_\cdot^{(j)}) - \boldsymbol{y}^{(j)}\|_1 & \dots \\ \dots & 0 & 0 & \dots & 0 & \hat{\delta}_C(\|f(\boldsymbol{x}_\cdot^{(j)}) - \boldsymbol{y}^{(j)}\|_1) & \dots \\ \dots & p & p & \dots & p & p & \dots \\ \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \dots \\ \dots & 1 & 1 & \dots & 1 & 1 & \dots \\ \dots & \boldsymbol{p}_{jn+1} & \boldsymbol{p}_{jn+2} & \dots & \boldsymbol{p}_{jn+n-1} & \boldsymbol{p}_{jn+n} & \dots \\ \dots & \boldsymbol{p}_{jn+1-s} & \boldsymbol{p}_{jn+2-s} & \dots & \boldsymbol{p}_{jn+n-1-s} & \boldsymbol{p}_{jn+n-s} & \dots \end{bmatrix} \tag{28}$$

However, with causal masking, we can only guarantee that p will appear in the columns containing the most recent example being queried. Thankfully, this is all that is needed. □

E.3. Task Execution

Lemma 4. *A two layer transformer, with embedding dimension $\mathcal{O}(d + \log(mn))$ can perform a task and weight its output by the proportion of examples of that task seen within the context.*

Now that the proportions of each task have been identified in the context, the task itself needs to be executed for the new example being queried. To simplify notation, let the input to this step be

$$\boldsymbol{X} = \begin{bmatrix} \dots & \boldsymbol{x}_1^{(m)} & \boldsymbol{x}_2^{(m)} & \dots & \boldsymbol{x}_{n-2}^{(m)} & \mathbf{=} \\ \dots & * & * & \dots & * & * \\ \dots & p & p & \dots & p & p \\ \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \dots & * & * & \dots & * & * \\ \dots & 1 & 1 & \dots & 1 & 1 \end{bmatrix} \tag{29}$$

Following the same process as outlined above, although with slightly different positional encodings, calculate $f(\boldsymbol{x}^{(m)})$ and place that result in the final column being decoded. These need to be added at the beginning of the construction, but are only introduced here for clarity.

$$\begin{bmatrix} \dots & \boldsymbol{x}_1^{(m)} & \boldsymbol{x}_2^{(m)} & \dots & \boldsymbol{x}_{n-2}^{(m)} & \boldsymbol{x}_{n-1}^{(m)} \\ \dots & * & * & \dots & * & * \\ \dots & p & p & \dots & p & p \\ \dots & * & * & \dots & * & f(\boldsymbol{x}^{(m)}) \\ \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \dots & * & * & \dots & * & * \\ \dots & 1 & 1 & \dots & 1 & 1 \\ \dots & 0 & 0 & \dots & 0 & 1 \\ \dots & 0 & 0 & \dots & 1 & 0 \end{bmatrix} \tag{30}$$

We will transform the row containing all p to be able to approximately multiply p by $f(\boldsymbol{x}^{(m)})$. Using the second to last row, perform $p \to 1 - p$. Using the last two rows, clear out the rest of that row and fill it with $-C$ for some large constant C. We then have

$$\begin{bmatrix} \dots & \boldsymbol{x}_1^{(m)} & \boldsymbol{x}_2^{(m)} & \dots & \boldsymbol{x}_{n-2}^{(m)} & \boldsymbol{x}_{n-1}^{(m)} \\ \dots & * & * & \dots & * & * \\ \dots & -C & -C & \dots & p & 1-p \\ \dots & * & * & \dots & * & f(\boldsymbol{x}^{(m)}) \\ \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \dots & * & * & \dots & * & * \\ \dots & 1 & 1 & \dots & 1 & 1 \\ \dots & 0 & 0 & \dots & 0 & 1 \\ \dots & 0 & 0 & \dots & 1 & 0 \end{bmatrix} \tag{31}$$

Further, use the second-to-last row to clear out all $*$ in the rows below.

$$\begin{bmatrix} \dots & \boldsymbol{x}_1^{(m)} & \boldsymbol{x}_2^{(m)} & \dots & \boldsymbol{x}_{n-2}^{(m)} & \boldsymbol{x}_{n-1}^{(m)} \\ \dots & * & * & \dots & * & * \\ \dots & -C & -C & \dots & p & 1-p \\ \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & f(\boldsymbol{x}^{(m)}) \\ \dots & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \dots & * & * & \dots & * & * \\ \dots & 1 & 1 & \dots & 1 & 1 \\ \dots & 0 & 0 & \dots & 0 & 1 \\ \dots & 0 & 0 & \dots & 1 & 0 \end{bmatrix} \tag{32}$$

These previous operations can all be done in a single MLP.

Lastly, use an attention layer where $\boldsymbol{W}_K$ selects the row with the $-C$s, $\boldsymbol{W}_Q$ selects the row with all 1s, and $\boldsymbol{W}_V$ selects the $f(\boldsymbol{x}^{(m)})$. For the last token $\boldsymbol{x}_L$,

$$\boldsymbol{X}^\top \boldsymbol{W}_K^\top \boldsymbol{W}_Q \boldsymbol{x}_L = \begin{bmatrix} \vdots \\ -C \\ -C \\ \vdots \\ p \\ 1-p \end{bmatrix} [1] = \begin{bmatrix} \vdots \\ -C \\ -C \\ \vdots \\ p \\ 1-p \end{bmatrix} \tag{33}$$

$$\sigma_S(\boldsymbol{X}^\top \boldsymbol{W}_K^\top \boldsymbol{W}_Q \boldsymbol{x}_L) \approx \begin{bmatrix} \vdots \\ -\infty \\ -\infty \\ \vdots \\ p \\ 1-p \end{bmatrix} = \begin{bmatrix} \vdots \\ 0 \\ 0 \\ \vdots \\ \frac{1}{1+e^{1-2p}} \\ 1 - \frac{1}{1+e^{1-2p}} \end{bmatrix} \tag{34}$$

$$\boldsymbol{W}_V \boldsymbol{x}_L \sigma_S(\boldsymbol{X}^\top \boldsymbol{W}_K^\top \boldsymbol{W}_Q \boldsymbol{x}_L) = \begin{bmatrix} \vdots \\ \mathbf{0} \\ \frac{1}{1+e^{1-2p}}\mathbf{0} + (1 - \frac{1}{1+e^{1-2p}}) f(\boldsymbol{x}^{(m)}) \\ \mathbf{0} \\ \vdots \end{bmatrix} \tag{35}$$

$$\boldsymbol{x}_L + \boldsymbol{W}_V \boldsymbol{x}_L \sigma_S(\boldsymbol{X}^\top \boldsymbol{W}_K^\top \boldsymbol{W}_Q \boldsymbol{x}_L) = \begin{bmatrix} \boldsymbol{x}_{n-1}^{(m)} \\ * \\ 1-p \\ \frac{1}{1+e^{1-2p}} f(\boldsymbol{x}^{(m)}) \\ \mathbf{0} \\ * \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{36}$$

Importantly, we are left with $\frac{1}{1+e^{1-2p}} f(\boldsymbol{x}^{(m)})$. The factor $\frac{1}{1+e^{1-2p}}$ is approximately p, especially around $\frac{1}{2}$. This multiplication can also be calculated more accurately with approximations using ReLUs or sigmoids, but for brevity and following experimental evidence of a sigmoid shape in task superpositions, these options are ommited.

E.4. Superposed Tasks with Parallel Heads

The above construction works for a single task, where the output is weighted by the proportions of the task within the context. To complete the construction of a transformer that does superposition of tasks, each of these models needs to be placed within the same overall transformer. This is described here.

Let there be a collection of tasks $\{t_i\}_{i=1}^T$ which can be executed by transformers with model weights represented by subscripts $(\cdot_i)$. With the input to each transformer being $\boldsymbol{X}^{(i)}$, the overall input matrix is given by vertically stacking these matrices.

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{X}_1 \\ \boldsymbol{X}_2 \\ \vdots \\ \boldsymbol{X}_{T-1} \\ \boldsymbol{X}_T \end{bmatrix} \tag{37}$$

Similarly, define each MLP's weights and biases as

$$\boldsymbol{W} = \text{diag}(\boldsymbol{W}_1, \ldots, \boldsymbol{W}_T) \quad \boldsymbol{b} = \begin{bmatrix} \boldsymbol{b}_1 \\ \vdots \\ \boldsymbol{b}_T \end{bmatrix} \tag{38}$$

This puts every MLP to be independent of each other. Lastly, we need to change the attention layers. This requires the use of one head per task. In each of the following, $\boldsymbol{W}^{(i)}$ is a weight matrix for head i, $(\boldsymbol{W})_i$ is the weight matrix for task i in its individual transformer, and each matrix below is in the i-th block.

$$\boldsymbol{W}_V^{(i)} = \begin{bmatrix} \vdots \\ \mathbf{0} \\ (\boldsymbol{W}_V)_i \\ \mathbf{0} \\ \vdots \end{bmatrix}^\top \quad \boldsymbol{W}_K^{(i)} = \begin{bmatrix} \vdots \\ \mathbf{0} \\ (\boldsymbol{W}_K)_i \\ \mathbf{0} \\ \vdots \end{bmatrix}^\top \quad \boldsymbol{W}_Q^{(i)} = \begin{bmatrix} \vdots \\ \mathbf{0} \\ (\boldsymbol{W}_Q)_i \\ \mathbf{0} \\ \vdots \end{bmatrix}^\top \tag{39}$$

In all, this model executes multiple tasks in superposition by using parallel streams of heads that each performs a single task. Task identification can happen through the same mechanism as task execution by comparing the output of the task on each in context example with the true output.

For context related tasks, there needs to be positional encodings that allow for looking back a fixed number of tokens. For context agnostic tasks, a wide MLP can be used to approximate arbitrary non-linear transformations of the input. Each of these tasks only require a small number of layers, significantly smaller than those of modern LLMs. It may be possible that LLMs do certain tasks with different combinations of layers.

Also, if we take the feature p from each parallel stream, this creates the following task identifier.

$$\boldsymbol{v} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_T \end{bmatrix} \tag{40}$$

Interpolating between the pure tasks, represented by unit vectors, different amounts of each task will appear in the superposition in roughly equal proportions to those found in $\boldsymbol{v}$.

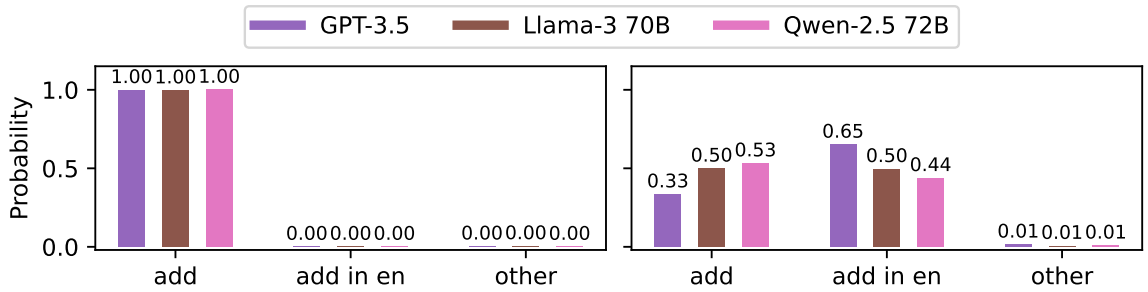
Lastly, we restate this construction formally.

Theorem 1. *A seven layer transformer with embedding dimension $\mathcal{O}(d + \log(mn))$ with K heads per attention layer can perform k tasks on vectors of dimension d in superposition, with weighting based on m different in-context examples each of length n .*

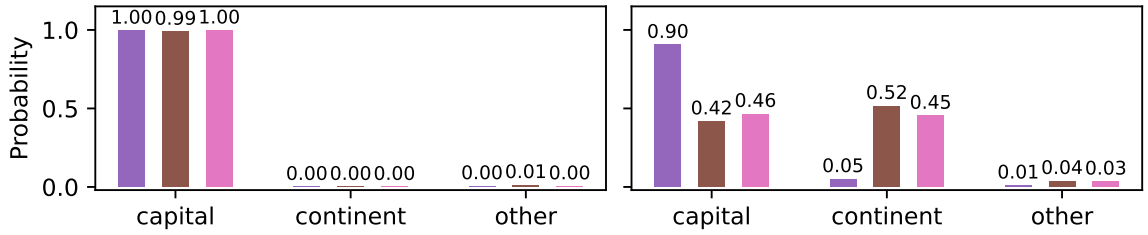
Proof. Using in succession each of Lemma 1, Lemma 2, Lemma 3, and Lemma 4, a transformer with the desired properties can execute k tasks in parallel. Lemma 1 identifies positions within the context that contain the labels $\boldsymbol{y}$. Lemma 2 then uses function approximation to perform arbitrary tasks within the architecture, which are then used by 3 to find the proportions of each task and aggregate them into a single task identifier. Lastly, Lemma 4 uses this task identifier to create a weighted sum of outputs from the different tasks based on their in-context proportions. □

Remark. *Transformers of greater depth than seven layers can also represent this construction by setting the weights in all other layers for the non residual part to zero.*

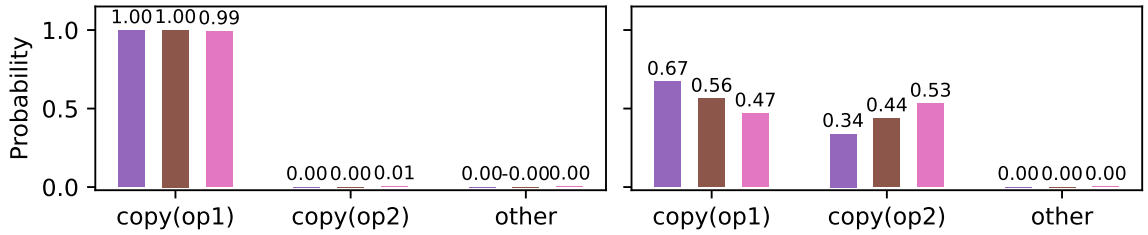
F. Additional experiment and analysis of task superposition in pretrained models



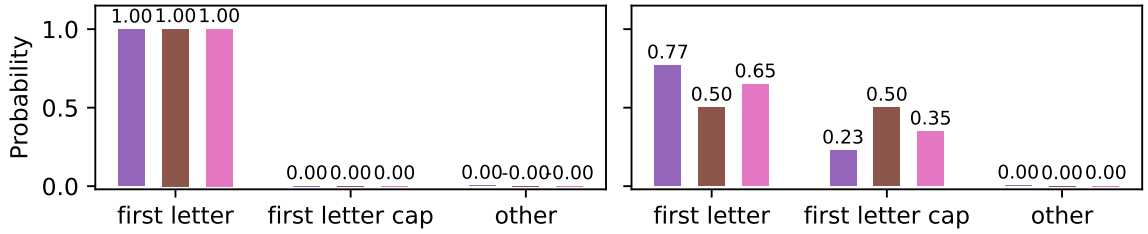
(a) **(left)** All task examples from task `add`; **(right)** equal number of task examples from `add` and `add_in_en`.



(b) **(left)** All task examples from task `capital`; **(right)** equal number of task examples from `capital` and `continent`.



(c) **(left)** All task examples from task `copy(op1)`; **(right)** equal number of task examples from `copy(op1)` and `copy(op2)`.



(d) **(left)** All task examples from task `first_letter`; **(right)** equal number of task examples from `first_letter` and `first_letter_cap`.

Figure 7: For each subplot, we consider two tasks `task1` and `task2` and on the left side we plot the median probability for task answers where all task examples in prompts are from `task1`; on the right side we plot the median probability for task answer where task examples from `task1` and from `task2` in prompts are equal.

F.1. Is superposition really happening?

In Figure 2 we plot distributions of probabilities for each task in four settings. Note that in setting 3 (corresponds to Figure 2b(left)), the `other` category has low probabilities for all models but for settings 1, 2 and 4 (correspond to Figure 2a(left), 2a(right) and 2b(right) respectively), the `other` category is not always low. A natural question to ask is:

> *For the non-zero probabilities we observe on each task answer, are they indications of task superposition or by-products of prediction noise?*

We set up an experiment to investigate this. For each setting, we select two tasks `task1` and `task2`; then we consider two scenarios: (1) we provide prompts where all task examples come from `task1` and (2) we provide prompts where half of the task examples come from `task1` and the other half of task examples come from `task2`; in both scenarios we measure the probabilities for task answers of `task1` and `task2` and see how these probabilities change between scenario (1) and (2). For each scenario, we test it on 300 prompts (each task has 10 in-context examples) and plot the median of task answers in Figure 7. As is shown in Figure 7 left side, where there is no task example from `task2` in the prompt, the probabilities for task answers of `task2` are near 0 for all models; on the right side, where there is an equal number of task examples that are from `task1` and `task2`, the probabilities for task answers of `task2` increase significantly. This indicates that when we provide prompts that mix task examples from different tasks, the prediction on each task answer is more than just pure prediction noise.

F.2. Measuring accuracy in task superposition

We further investigate how task superposition affect task performance. In particular, for a prompt consisting examples of K tasks (each task has an equal number of task examples), we define a task being correctly performed if its task answer lies in the top-K answers (that we use beam search to find). We compare the accuracy against individual task accuracy where we provide prompts consisting of task examples of only 1 task and define the task being correctly performed if the task answer is the top-1 answer.

We calculate accuracy in $K = 1$ case and $K > 1$ case using 300 prompts and show the result in Table 2. We find that as we increase the number of tasks, all models exhibit an accuracy decrease in correctly performing each individual task.

Model	$K = 1$				$K = 4$			
	t1	t2	t3	t4	t1	t2	t3	t4
GPT-3.5	100	99.7	90	95	97.3 (−2.7)	90.7 (−9)	80 (−10)	80.3 (−14.7)
Llama-3 70B	100	100	97	99	100 (0)	99 (−1)	95.7 (−1.3)	93 (−6)
Qwen-2.5 72B	100	100	87	96.7	99.7 (−0.3)	87.7 (−12.3)	87.7 (+0.7)	63.7 (−33)

(a) Setting 1: Addition in original numerical form and in different languages where `t1` = `add`, `t2` = `add_in_en`, `t3` = `add_in_fr`, `t4` = `add_in_es`.

Model	$K = 1$			$K = 3$		
	t1	t2	t3	t1	t2	t3
GPT-3.5	97.7	100	100	93 (−4.7)	61.7 (−38.3)	42 (−58)
Llama-3 70B	98.3	100	100	85.3 (−13)	79 (−21)	75 (−25)
Qwen-2.5 72B	98.3	100	100	87.7 (−10.6)	78.3 (−21.7)	30 (−70)

(b) Setting 2: Naming the capital, continent and capitalize the country name where `t1` = `capital`, `t2` = `continent`, `t3` = `capitalization`.

Model	$K = 1$			$K = 3$		
	t1	t2	t3	t1	t2	t3
GPT-3.5	100	100	100	100 (0)	98.3 (−1.7)	97.3 (−2.7)
Llama-3 70B	100	100	99.7	99.7 (−0.3)	99.7 (−0.3)	99.3 (−0.4)
Qwen-2.5 72B	100	100	100	100 (0)	100 (0)	99 (−1)

(c) Setting 3: `t1` = `copy(op1)`, `t2` = `copy(op2)` and `t3` = `op1+op2`.

Model	$K = 1$				$K = 4$			
	t1	t2	t3	t4	t1	t2	t3	t4
GPT-3.5	100	100	100	97	62 (−38)	79.7 (−20.3)	79 (−21)	24.3 (−72.7)
Llama-3 70B	100	85	100	59.3	85 (−15)	52.7 (−32.3)	95.3 (−4.7)	39 (−20.3)
Qwen-2.5 72B	100	98.7	100	97	53.3 (−46.7)	98.3 (−0.4)	42 (−58)	84 (−13)

(d) Setting 4: First or last letter in upper or lower cases where `t1` = `first_letter`, `t2` = `last_letter`, `t3` = `first_letter_cap`, `t4` = `last_letter_cap`.

Table 2: Accuracy for each task in percentage, with the delta change given in parenthesis. For each setting we calculate the accuracy with prompts consisting of task examples of only one task ($K = 1$ case) and with prompts consisting of examples from multiple tasks ($K > 1$ case).