Self-MoE: Self Mixture of Experts in between decoder layers

Anonymous EMNLP submission

Abstract

Mixture of Experts is a well-known technique in machine learning and is widely used to empower large language models. Unfortunately, it requires a lot of resources to train experts. To weaken this requirement, we propose a modification to the architecture of pretrained LLMs we call self Mixture of Experts (self-MoE), which is a mixture of experts with all the experts being the same exact model. This adjustment adds a handful of weights and yields a significant improvement in model performance. We evaluated self-MoE on mathematical reasoning and code generation and observed significant improvements across various benchmarks. We plan to publish the training code and the model weights upon acceptance.

1 Introduction

Over the past three years, the field of large language models has experienced rapid advancements with the introduction of the Generative Pre-trained Transformer (GPT) [3]. These models have demonstrated exceptional performance across a wide range of tasks, capturing the attention of researchers and practitioners worldwide. A significant leap in the development of large language models was marked by the emergence of other models such as Gemini, Llama, and Mistral, which aimed to build upon the achievements of the GPT series.

Mistral AI, a company specializing in language models, took a novel approach with the release of their latest model Mixtral [15]. This is a sparse mixture-of-experts network that combines multiple Mistral models, each trained on specific domains, into a single unified model. Mixtral uses a feed-forward block as a router network, which selects two of eight groups to process each token and combines their outputs, increasing model parameters while controlling cost and latency. This results in a model that performs exceptionally well across multiple tasks, providing a more comprehensive and versatile solution for a wide array of applications.

Our proposed idea is inspired by two recent published works, the aforementioned Mixtral of Experts [15] and LASER [26] papers. The first paper demonstrates that the incorporation of multiple MLP layers to the decoder in the model's architecture can significantly improve its ability to handle a wide range of tasks. By allowing each decoder layer to contribute to the final output, the model can better adapt to different input domains, leading to improved performance and generalization capabilities. The LASER paper highlights a potential issue in LLMs, where information from earlier decoder layers can be forgotten or distorted as the model progresses through subsequent layers.

We propose a novel addition to the LLMs architectures called self Mixture of Experts (self-MoE). This method combines the output of each decoder layer with the output of the previous decoding layer through additional gates. By merging each two sequential decoder layers as two experts from a single model, we aim to improve the performance of the model while reducing its complexity. The use of gates enables the model to selectively merge the outputs of each decoder layer, creating a more dynamic and adaptive system.

We evaluate our method on code generation and natural language mathematics benchmarks and demonstrate significant quality boost for self-MoE compared to baselines. In general, our contributions can be summarized into following:

1. We propose self-MoE, a simple and effective approach to improve pretrained language model performance.

2. We present a thorough investigation of various gate types and demonstrate their impact on the evaluation benchmarks, while also introducing a method for gates positioning selection.

3. We show that our method achieves improved results on diverse benchmarks, including math problem solving and code generation.

2 Related work

Routing Networks. In routing networks input tokens are transformed by dynamically engaging with a select subset of network parameters. This characteristic is commonly found in sparsely-activated networks, but also applies to networks featuring early exits. This concept resembles out gating mechanism, since both methods are skipping intermediate layers, but instead of routing we use averaging.

In papers exploring **early exits**, models learn to determine the optimal point to terminate computation, enabling the token to bypass subsequent transformer layers. This idea first emerged in CNNs [29], where branch modules are inserted at various exit points within a deep learning network. In [13, 20] authors explored idea of anytime predictions in convolutional networks, which reduced total computation. Then early exiting was explored in language models for both its inference [25, 8], where it is proposed to use early exit loss or local confidence measures to speed-up inference, and training [7, 19], where dedicated LM head was added for each decoder layer in an encoder-decoder model. In [24] routers are used to choose among potential computational paths. This approach shares similarities with ours, but differs in its use of probabilistic routing mechanisms, whereas we employ summation. Moreover, our primary focus is on improving the quality of generations, rather than optimizing computational efficiency.

Mixture of Experts was explored in [6] as a component of deeper networks, enabling large and efficient models. Idea of dynamic component activation based on input tokens led to [27] where the idea was scaled to a 137B LSTM by introducing sparsity, achieving fast inference at high scale. However, this work faced challenges like high communication costs and training instabilities. Currently in the usual sense MoE consists of two main components: sparse MoE layers that replace traditional dense feed-forward network layers, and a gate network or router that determines which tokens are sent to which expert. Mixtral proposed in [15] uses router network which selects two experts to process the current state and combine their outputs for each token. In such method information is extracted from decoders broadwise, leveraging parallel outputs from multiple experts; in contrast, our approach involves extracting information from decoders by going in depth, combining outputs from sequential decoders.

2.1 Knowledge from earlier layers

Language models have been observed to suffer from forgetting, where knowledge acquired in earlier layers is lost as the model progresses to later layers [26]. Recent studies shows that specific knowledge is preserved in intermediate layers, for example space and time representations [11] are learned and stored across multiple scales. Experiments in [17] reveal that LLMs encode more context knowledge in upper layers, initially focus on knowledge-related entity tokens in lower layers, and gradually forget earlier context knowledge in intermediate layers when presented with irrelevant evidence. Results from [16] show that simpler tasks can be probed in shallow layers, while more complex tasks require deeper layers for accurate understanding. The presented works suggest that earlier layers of the model often contain information that would be useful for forming the final generation.

3 Method

In this section, we briefly discuss the latest Large Language Models (LLMs) architectures, focusing on Llama and Mistral. We present additional gates as addition to transformer acrhitecture and explain how to build a self-MoE model and utilize it. Lastly, we examine various types of gates that we've explored in our research.

3.1 LLMs review

The LLMs architectures we examined are vast decoder-only transformer models with billions of parameters. These models are trained on extensive datasets, containing billions or even trillions of tokens [23]. The design of these models encompasses an embedding layer that transforms each token into a latent vector. These vectors are then processed through decoder layers to forecast the succeeding token. This process is repeated (autoregressively) to produce the final answer. Each transformer decoder layer comprises a self-attention layer, an MLP block, and a normalization layer [31]. The model concludes with a head layer that translates the output embedding vector into the distributed probabilities of the upcoming token. In essence, the primary distinction between the Llama [30] and Mistral [14] models lies in their attention mechanisms. While both models share a highly similar overall architecture, Mistral distinguishes itself by employing grouped query attention (GQA) and sliding window attention.

3.2 self-MoE

In essence, the self-MoE concept is an extension of the traditional Mixture of Experts (MoE) model. However, rather than combining different models in the MLP section, self-MoE merges decoder layers. This is achieved by incorporating gates between them, leading to a minimal increase in the number of training parameters - approximately 200M for 7B models. Fig. 1 shows the integration of the gates into the LLMs architecture as we mentioned above.

Assuming we have a standard transformer decoder model consisting of n blocks, the conventional large language model (LLM) during forward pass at each transforms the the input n times by sequentially applying blocks on previous outputs:

$$h_i = B_i(h_{i-1}). \quad (1)$$

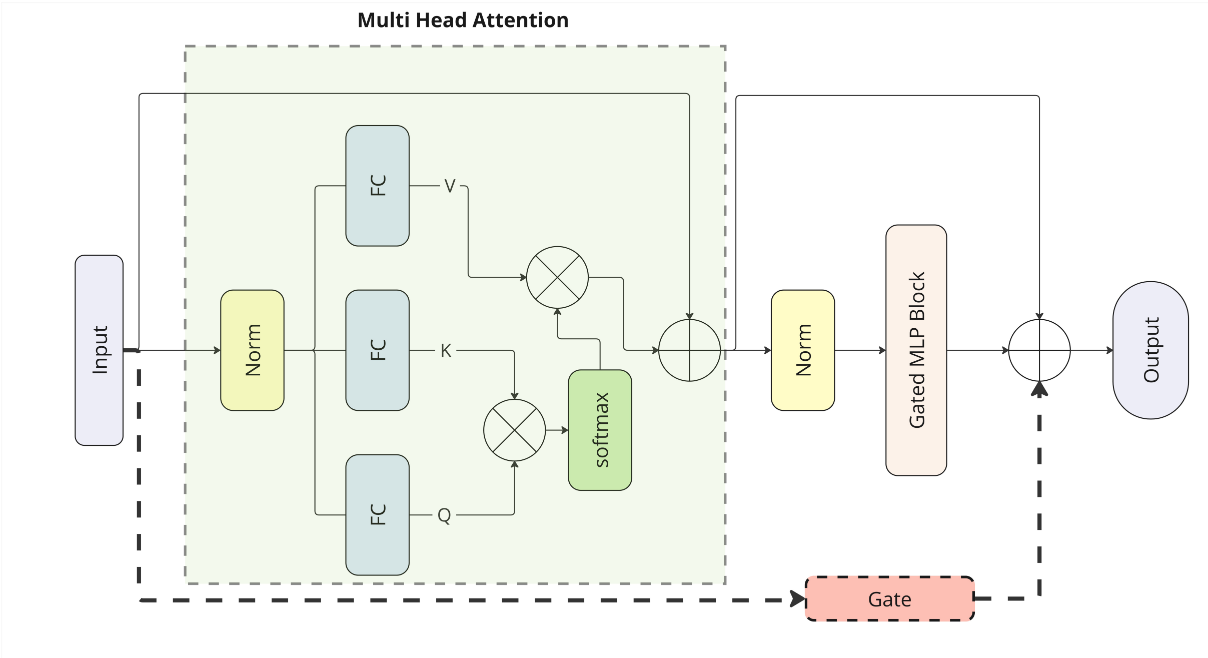

Figure 1: The figure schematically shows the architecture of a modified transformer decoder block. The proposed adjustment is an additional trainable gate, which is a direct connection between the input and output of a decoder. In section 3.3 we demonstrate how gates differ from skip-connections.

Within each block, there are self-attention and MLP blocks with skip connections.

Self-MoE introduces an additional G gate layer that takes the output of the previous decoder layer and combines it with the output of the current decoder layer as follows:

$$h_i = B_i(h_{i-1}) + G_i(h_{i-1}). \tag{2}$$

When it comes to applying this concept to an LLM with n layers, it necessitates the use of $n-1$ gates, following the same approach. There are two primary paths to pursue from this point. The first is to concentrate on accelerating the model's inference by enabling it to select a single path, either through the gate or the subsequent decoder layer, rather than employing both. However, this would necessitate training the initial model from scratch with the gates or, at the very least, on a substantial amount of data. Our primary objective, though, is to achieve performance comparable to Mixtral with a minimal increase in the number of parameters and nearly the same latency as the base 7B model. To accomplish this, we initialize the Gates' weights to zero, causing 2 to revert to the base model's 1 at the beginning of the training.

3.3 Gates

In this section, we will discuss two distinct subjects. Firstly, we will examine the types of gates utilized in our system. Subsequently, we will illustrate the process of determining which gates to retain or discard following the training phase.

Gates type We experimented with three distinct gate types to demonstrate that the self-MoE concept is not solely dependent on the additional skip connection, but also on the specific type of gate utilized.

1. Matrix Linear Gate Layer: The input dimension of this layer corresponds to the input dimension of the decoder layer, while the output dimension is equivalent to the output dimension of the decoder.

2. Vector Linear Gate Layer: Each channel of the input dimension in this layer has a single value that is allowed to pass through.

3. Single value Gate layer: The output of the previous layer is multiplied element-wise by a single trainable parameter and then added to the output of the subsequent layer.

Gates selection After training we can eliminate certain gates that do not significantly contribute to model performance. To accomplish this, we employ a simple yet effective approach of evaluating the changes in perplexity throughout the layers for a portion of the training set (calibration data). We utilize language model head as a projection layer to map each hidden state, both before and after incorporating the gates' output, into the output space. Subsequently, we compute the perplexity for

Model	GSM8k	MATH	MMLU-STEM
mistralai/Mistral-7B-v0.1	0.378	0.129	-
w/o self-MoE	0.398	**0.147**	0.65
with self-MoE	**0.419**	0.146	**0.67**
mistralai/Mistral-7B-Instruct-v0.2	0.400	0.103	-
w/o self-MoE	0.410	0.123	0.60
with self-MoE	**0.431**	**0.125**	0.60
meta-llama/Meta-Llama-3-8B	0.458	0.150	-
w/o self-MoE	0.434	0.150	0.61
with self-MoE	**0.476**	**0.167**	**0.64**
microsoft/Phi-3-mini-128k-instruct	0.695	0.224	-
w/o self-MoE	0.699	0.265	0.62
with self-MoE	**0.700**	**0.269**	0.62

Table 1: Comparison of fully trained base models and same models trained with self-MoE. Our method demonstrates quality improvements across all baselines. For the base Meta-Llama-3-8B, full training degrades its mathematical problem-solving abilities, whereas self-MoE provides a significant boost.

both scenarios and determine the average across all calibration data. Finally, we discard the gates that exhibit minor changes in perplexity values. In other words, assume that we have a two hidden states h_t and h_{t-1}, let us denote to the gates as G function, and the lm_head as $proj$. For an input x we calculate the perplexity $perp$ as follows:

$$p_{t1} = \mathrm{perp}(\mathrm{proj}(h_t)_x, x), \tag{3}$$

where $proj(h_t)_x$ is the projection of the hidden state at layer number t of an input x, p_{t1} is the perplexity value without gates at layer t.

$$p_{t2} = \mathrm{perp}(\mathrm{proj}(h_t + G(h_{t-1}))_x, x). \tag{4}$$

Then we decide to keep the gates or drop as follows for each decoder layer t:

$$G_t = \begin{cases} 0, & \text{if } p_{t2} \leq p_{t1} \\ G_t, & \text{otherwise.} \end{cases} \tag{5}$$

Gates vs skip connections Let us dive into the changes through the hidden state h_{i-1} as an input to a decoder layer. At the first attention part Att we can define the intermediate output A as follows:

$$A = \mathrm{Att}(\mathrm{norm}_1(h_{i-1})) + h_{i-1}. \tag{6}$$

We pass the attention output to the MLP layer mlp with another skip connection so the current hidden state h_i is calculated as follows:

$$h_i = \mathrm{mlp}(\mathrm{norm}_2(A)) + A, \tag{7}$$

substituting 6 in 7 we get:

$$h_i = \mathrm{mlp}(\mathrm{norm}_2(A)) + \mathrm{Att}(\mathrm{norm}_1(h_{i-1})) + h_{i-1}. \tag{8}$$

It is obvious that adding a skip connection will not have any impact. From 2 and 8, we can formulate self-MoE as follows:

$$h_i = \mathrm{mlp}(\mathrm{norm}_2(A)) + \mathrm{Att}(\mathrm{norm}_1(h_{i-1})) + h_{i-1} + G_i(h_{i-1}). \tag{9}$$

The self-MoE can be perceived as a trainable linear selector that assigns weights to the previously obtained information. This approach offers two primary advantages:

1. When the weighting values are high, it can be considered as two experts collaborating to generate the output, similar to the MoE method.
2. When the weighting values are low, it can be perceived as a trainable Laser method.

4 Experiments

To assess the efficiency of the self-MoE we do full-model training of various architectures across a range of tasks. We focus on two domains while comparing our proposed architectures with base transformer models trained on specific data. The two tasks are mathematical problem-solving and code generation.

4.1 Math Evaluation

The first domain is mathematical problem-solving. We use benchmarks which are commonly used for evaluating mathematical and logical abilities in language models [2, 18]: GSM8K [5] - one of the benchmarks listed on the Open LLM Leaderboard, MMLU-STEM - a subset of STEM subjects defined in MMLU, and MATH, consisting of challenging competition mathematics problems [12].

Our goal is to test the hypothesis that the adding of gates enhances the model's task understanding, independent of its architecture. To validate this assertion, we select a diverse set of baselines of varying size, which is used for evaluation. During evaluation we minimize the base model capabilities

Model	Params	Python	C++	Java	JS	Go	AVG
deepseek-coder-1.3b	1.3B	33.8	20.2	30.7	28.3	30.1	28.62
w/o self-MoE	1.3B	34.1	23.5	31.7	27.8	30.3	29.48
with self-MoE	1.4B	**37.3**	**24.1**	**33.1**	**31.6**	**30.9**	**31.40**
deepseek-coder-6.7b	6.7B	40.1	30.3	38.7	35.6	39.8	36.90
w/o self-MoE	6.7B	43.6	31.5	40.0	34.2	39.3	37.72
with self-MoE	7.3B	**44.2**	**32.7**	**41.8**	33.3	38.7	**38.14**

Table 2: Pass@1 scores for base models and self-MoE models across different programming languages in HumanEval-X. For smaller deepseek model training with self-MoE enhances code generation for each of the six selected programming languages. For model with 6.7B parameters, the baseline model shows better results for JavaScript and Go.

Model	Params	GSM8k	MATH	MMLU-STEM
Meta-Llama-3-8B	8B	0.458	0.150	-
Matrix Linear Gate	8.5B	0.476	**0.170**	**0.64**
Vector Linear Gate	8B	**0.489**	0.161	0.62
Single value Gate	8B	**0.489**	0.162	0.62

Table 3: Comparison of different gate types for meta-llama/Meta-Llama-3-8B model on math benchmarks. Vector Linear and Single value Gate types demonstrate a significant improvement in quality on the GSM8K benchmark, despite having nearly as many parameters as the baseline model.

in a particular task, which allows us to isolate the impact of fine-tuning and architectural features on task performance. We deliberately choose models with a low likelihood of having mathematical data from the benchmarks in their training sets. Notably, our selection of models comprises some of the most recent and compact open-source models, including Mistral-7B-v0.1, Meta-Llama-3-8B, Phi-3-mini-128k-instruct, and Mistral-7B-Instruct-v0.2, which represent the latest top-performing models in their class.

Models are trained on Orca-Math [22], which is a high quality synthetic dataset of 200K math problems. We choose to train models with Matrix Linear Gates. All models are trained on A100 40Gb GPUs, using a combination of data and model parallelism. Models are trained with a context length of 1024, a batch size of 1, and gradient accumulation of 64, spanning 128000 iterations ($\sim$2 epoches). Optimization is performed with AdamW [21] with a learning rate of 2.5×10^{-6} and 12800 warmup steps.

Results are provided in Table 1. The self-MoE improves the performance of all base models, particularly the Mistral models. Our method significantly enhances the model's abilities compared to models trained without gates.More specifically, the improvement on GSM8K benchmark for Mistral is 10% for 0.5B additional parameters.

4.2 Code Evaluation

For evaluating the effectiveness of the self-MoE approach in code generation tasks, we utilized the HumanEval-X [32] benchmark. This benchmark is designed to assess the multilingual capabilities of code generation models by focusing on the functional correctness of the generated programs, rather than just semantic similarity.

HumanEval-X comprises 820 high-quality human-crafted samples, each accompanied by test cases. The benchmark covers multiple programming languages, including Python, C++, Java, JavaScript, and Go. Each sample in HumanEval-X consists of a declaration, docstring, and solution, which can be combined in various ways to support different downstream tasks such as code generation. The model uses the declaration and docstring as input to generate the solution.

The primary metric used for evaluation is the unbiased pass@1 metric proposed in Codex [4], which measures the functional correctness of the generated code across multiple attempts.

We trained and evaluated DeepSeek models [10]. The training data was mined from GitHub and filtered using heuristics and a code embedding model to closely match the distribution of HumanEval-like data.

The filtering approach involves using a pretrained unsupervised CCT [28] encoder to select [1] relevant data by measuring vector similarity between embeddings of mined GitHub code and multilingual data from MBPP and similar benchmarks. This resulted in a dataset that is more aligned with the tasks presented in HumanEval-X and consist of 20k examples .

We observed improvements in code generation quality across most programming languages when

	Python	C++	Java	JS	Go
deepseek-coder-1.3b	33.8	20.2	30.7	28.3	30.1
Matrix Linear Gate	**37.3**	24.1	33.1	31.6	30.9
Vector Linear Gate	33.8	**24.7**	**35.7**	**32.8**	30.3
Single Value Gate	35.3	**24.7**	32.9	29.0	**31.5**

Table 4: Comparison of different gate types for deepseek-ai/deepseek-coder-1.3b model in HumanEval-X. On average, Vector Linear Gate provides the highest improvement to the metric values.

applying the self-MoE approach. The self-MoE models demonstrated better performance in terms of functional correctness compared to their base counterparts. This improvement can be attributed to the enhanced ability of the self-MoE architecture to capture and utilize information from earlier layers more effectively.

The results are summarized in Table 2, showing the pass@1 scores for different models and configurations. More specifically, the improvement on C++ benchmark for the 1.3B deepseek model is about 20% for 0.1B additional parameters.

Overall, the self-MoE approach consistently enhances the performance of the models, particularly in complex code generation tasks, demonstrating its efficacy in improving the quality of generated code.

4.3 Ablation Study

We find that adding self-MoE improves performance over the base model, therefore we investigate if we can further improve by changing the gates configuration. In this section we conduct an ablation study where we experiment with the type of gate and its selection, and find the most optimal self-MoE structure.

4.3.1 Gates Type

We compare different types of gates proposed in section 3.3 on both math solving (Table 3.3) and code generation (Table 4) tasks. Our results show that no single type of gate consistently outperforms others across all benchmarks. However, Vector Linear Gate and Single Value Gate have significantly fewer parameters, with approximately the same number as the base model. This suggests that these type of gates can be used to save resources without sacrificing performance.

4.3.2 MoE vs Self-MoE

In Table 5 we compare our method with original Mixture of Experts on mathematical problem-solving task. We pick the original Mistral-7B model and modify it as follows: the first model is enhanced with the self-MoE method, while the second is obtained by merging two identical base models using MergeKit [9] into an MoE model. Both versions are trained on the Orca-Math data with same parameters as described in section 4.1.

The comparison on all three benchmarks reveals a significant improvement in mathematical problem solving, with the self-MoE model achieving this improvement with a substantially smaller number of parameters (8.5B vs 13B).

4.3.3 Gates selection

Based on our earlier discussion in Section 3.3, we employ perplexity analysis across all layers to determine whether to retain or discard gate for each decoder layer in the model. By comparing the perplexity changes with and without gates across the layers we find that the disparity in perplexity becomes evident beyond the $18th$ layer for Mistral and $16th$ layer for Llama3. Therefore, we eliminate all Gates preceding these layers for models we trained with self-MoE. The outcomes of implementing our proposed method are depicted in Table 6.

4.4 Qualitative results

The proposed adjustment demonstrated enhanced performance across a diverse range of tasks, but it is equally crucial to evaluate its efficacy on fundamental mathematical tasks, as an example we show on Fig. 2 a comparison of solving a quadratic equation with two variables. we present the generated answers using three distinct Mistral models. The first model is trained with the proposed gates, the second model is a hybrid Mixtral-based model combining two Mistral models, and the third model is a Mistral model trained with additional gates.

5 Conclusion

We present the self-MoE method based on incorporating gates between decoder layers in LLM architecture and show its versatility across various base models. Proposed method demonstrate significant improvement in performance on different benchmarks in mathematical reasoning and code generation, while requiring minimal additional resources to train. Notably, our approach outperforms the Mixtral of Experts model, which requires training multiple distinct experts, thereby making it a more efficient and effective solution for empowering large language models.

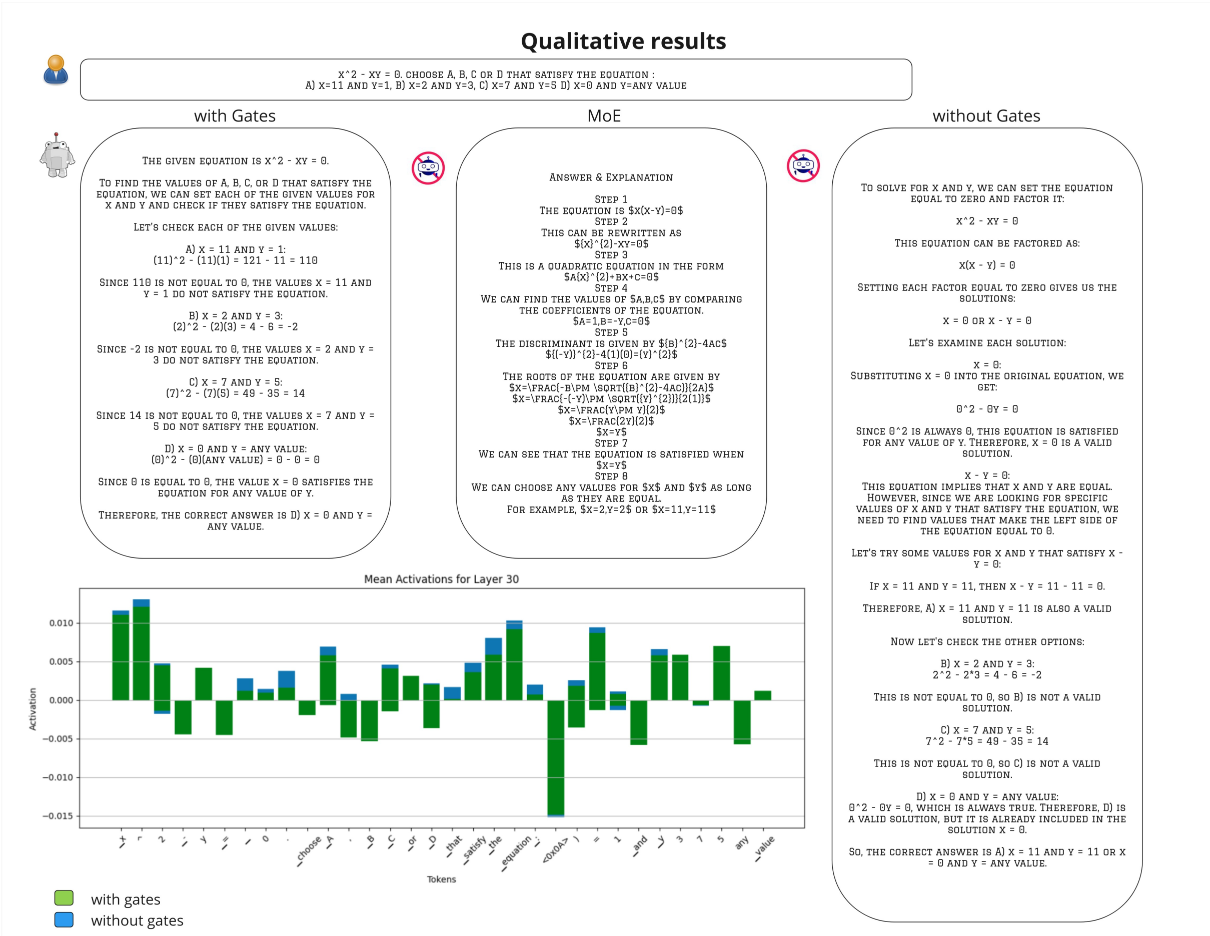

Figure 2: The figure showcases the performance of three distinct models when tested with a basic mathematical problem. The left-hand side displays the solution provided by a Mistral model equipped with the proposed gates. The middle section shows the response from a Mixtral model, which is comprised of two Mistrals. Lastly, the right-hand side showcases the answer given by a Mistral model that was trained without gates. The bottom left corner highlights the mean activation of the final layers for Mistral models, both with and without gates.

	GSM8k	MATH	MMLU-STEM
Mistral-7B-v0.1	0.378	0.129	-
MoE	0.407	0.135	0.65
self-MoE	**0.419**	**0.146**	**0.67**

Table 5: Comparison between MoE and self-MoE methods when trained on the same dataset. Our method outperforms on mathematical benchmarks, despite having nearly half the number of parameters compared to the Mixtral model.

Model	Gates	Selective Gates	GSM8k
Mistral-7B-Instruct-v0.2			0.410
Mistral-7B-Instruct-v0.2	✓		0.431
Mistral-7B-Instruct-v0.2	✓	✓	**0.437**
Meta-Llama-3-8B			0.458
Meta-Llama-3-8B	✓		0.476
Meta-Llama-3-8B	✓	✓	**0.480**

Table 6: The impact of gates and selective gates based on perplexity analysis. Removing gates from earlier decoders boosts models' performance.

Limitations

The methodology and experiments detailed in this paper have certain limitations. Notably, our method shows a significant improvement in performance only on specific tasks and we cannot guarantee that this advance will transfer entirely to more general tasks. Moreover adding gates to the model is associated with increase of model size and therefore higher resource consumption, however we provide various gate types where such increase is minimal.

References

[1] Dmitry Abulkhanov, Nikita Sorokin, Sergey Nikolenko, and Valentin Malykh. Lapca: Language-agnostic pretraining with cross-lingual alignment. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 2098–2102, New York, NY, USA, 2023. Association for Computing Machinery.

[2] Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*, 2023.

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

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[5] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[6] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. 2014.

[7] Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. 2020.

[8] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024.

[9] Charles Goddard, Shamane Siriwardhana, Malikeh Ehghaghi, Luke Meyers, Vlad Karpukhin, Brian Benedict, Mark McQuade, and Jacob Solawetz. Arcee's mergekit: A toolkit for merging large language models. *arXiv preprint arXiv:2403.13257*, 2024.

[10] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. Deepseek-coder: When the large language model meets programming – the rise of code intelligence, 2024.

[11] Wes Gurnee and Max Tegmark. Language models represent space and time. *arXiv preprint arXiv:2310.02207*, 2023.

[12] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. 2021.

[13] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q. Weinberger. Multi-scale dense networks for resource efficient image classification. 2018.

[14] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[15] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[16] Mingyu Jin, Qinkai Yu, Jingyuan Huang, Qingcheng Zeng, Zhenting Wang, Wenyue Hua, Haiyan Zhao, Kai Mei, Yanda Meng, Kaize Ding, et al. Exploring concept depth: How large language models acquire knowledge at different layers? *arXiv preprint arXiv:2404.07066*, 2024.

[17] Tianjie Ju, Weiwei Sun, Wei Du, Xinwei Yuan, Zhaochun Ren, and Gongshen Liu. How large language models encode context knowledge? a layer-wise probing study. *arXiv preprint arXiv:2402.16061*, 2024.

[18] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022.

[19] Yijin Liu, Fandong Meng, Jie Zhou, Yufeng Chen, and Jinan Xu. Faster depth-adaptive transformers. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 13424–13432. AAAI Press, 2021.

[20] Zhuang Liu, Zhiqiu Xu, Hung-Ju Wang, Trevor Darrell, and Evan Shelhamer. Anytime dense prediction with confidence adaptivity. 2022.

[21] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. 2019.

[22] Arindam Mitra, Hamed Khanpour, Corby Rosset, and Ahmed Awadallah. Orca-math: Unlocking the potential of slms in grade school math. *arXiv preprint arXiv:2402.14830*, 2024.

[23] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[24] David Raposo, Samuel Ritter, Blake A. Richards, Timothy P. Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *CoRR*, abs/2404.02258, 2024.

[25] Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472, 2022.

[26] Pratyusha Sharma, Jordan T Ash, and Dipendra Misra. The truth is in there: Improving reasoning with layer-selective rank reduction. In *The Twelfth International Conference on Learning Representations*, 2023.

[27] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. 2017.

[28] Nikita Sorokin, Dmitry Abulkhanov, Sergey Nikolenko, and Valentin Malykh. Cct-code: Cross-consistency training for multilingual clone detection and code search, 2023.

[29] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*, pages 2464–2469. IEEE, 2016.

[30] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[32] Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, Teng Su, Zhilin Yang, and Jie Tang. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x, 2023.