

Dense Communication between Language Models

Shiguang Wu¹ Yaqing Wang² Quanming Yao¹

¹Department of Electronic Engineering, Tsinghua University

²Beijing Institute of Mathematical Sciences and Applications

wsg23@mails.tsinghua.edu.cn

wangyaqing@bimsa.cn

qyaoaa@tsinghua.edu.cn

Abstract

As higher-level intelligence emerges from the combination of modular components with lower-level intelligence, many works combines Large Language Models (LLMs) for collective intelligence. Such combination is achieved by building communications among LLMs. While current systems primarily facilitate such communication through natural language, this paper proposes a novel paradigm of direct dense vector communication between LLMs. Our approach eliminates the unnecessary embedding and de-embedding steps when LLM interact with another, enabling more efficient information transfer, fully differentiable optimization pathways, and exploration of capabilities beyond human heuristics. We use such stripped LLMs as vertexes and optimizable seq2seq modules as edges to construct LMNet, with similar structure as MLPs. By utilizing smaller pre-trained LLMs as vertexes, we train a LMNet that achieves comparable performance with LLMs in similar size with only less than 0.1% training cost. This offers a new perspective on scaling for general intelligence rather than training a monolithic LLM from scratch. Besides, the proposed method can be used for other applications, like customizing LLM with limited data, showing its versatility.

1 Introduction

Large Language Models (LLMs) have achieved impressive performance in natural language understanding, generation, and reasoning [5]. Modern LLMs exhibit general intelligence capabilities across a wide range of subjects [1, 52, 11], but still face limitations when tackling complex tasks that require domain-specific expertise or collaborative reasoning.

Inspired by the Society of Mind theory [29], which suggests that higher-level intelligence can emerge from the coordination of simpler components, recent research has explored building collective intelligence by combining multiple LLMs [46, 10]. This is typically achieved by designing communication structures among LLMs, such as chain-of-thought prompting where a single model communicates with itself iteratively [49, 30, 57], or multi-agent systems where LLMs form a network interacting with each other and the environment [16, 58, 59]. Most of these systems use natural language as the medium of communication, due to the fact that LLMs are pre-trained on text

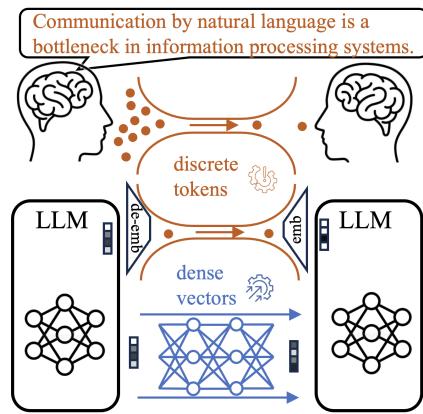


Figure 1: Communication between LLMs through dense vectors eliminates the bottleneck of natural language.

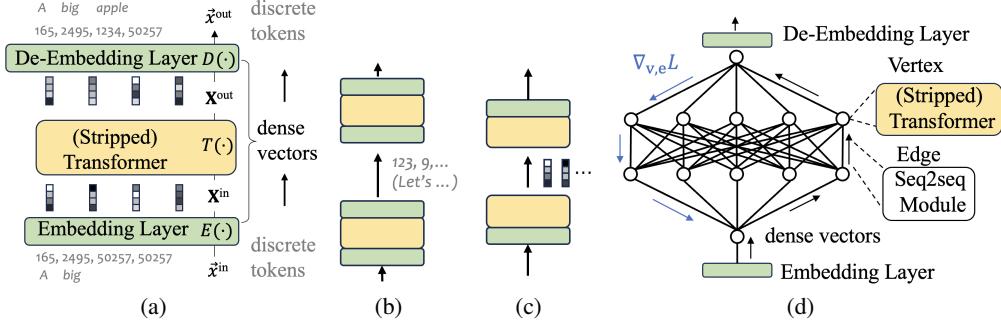


Figure 2: Illustration of the proposed paradigm. (a) A standard LLM processes discrete token inputs by embedding them into dense vectors, and outputs discrete tokens via a de-embedding layer. (b) Existing communication between LLMs typically occurs through discrete tokens. (c) Our approach strips the embedding and de-embedding layers, allowing LLMs to communicate directly via dense vectors. (d) We construct and train a LMNet by connecting stripped transformers with trainable communication modules.

inputs and outputs. Natural language, however, is a symbolic and discrete representation, originally designed to align human understanding [50, 7]. To be processed by LLMs, discrete tokens must be embedded into dense vectors before being passed through the model, and then decoded back into tokens at the output [3, 27, 45]. This introduces redundancy and inefficiency. As suggested by studies in cognitive science [28, 19, 32, 9], language is not an efficient way to transfer large amounts of information. While natural language is essential for human interaction and model pre-training, it is not necessarily suitable for communication between LLMs.

This paper proposes a new paradigm of dense vector communication between LLMs. Instead of relying on discrete tokens, one LLM can output internal hidden states as dense vectors, and another LLM can directly accept these vectors as input—bypassing the embedding and de-embedding steps entirely. This mechanism offers several key advantages:

- **Higher information efficiency:** Dense vectors carry richer information per token, reducing loss from embedding and de-embedding.
- **Fully differentiable communication:** The entire system becomes end-to-end differentiable, supporting gradient-based optimization.
- **Machine-native communication:** LLMs can learn roles and protocols suited for inter-model collaboration, unconstrained by human-designed language.

We instantiate this paradigm in a framework called LMNet, where each vertex is a stripped transformer without embedding or output layers, and each edge is a trainable sequence-to-sequence (seq2seq) module that transforms dense vectors between models. This structure is reminiscent of a multilayer perceptron, with LLMs acting as nonlinear modules connected by communication layers.

As a case study, we construct and train LMNet-1B, using 0.5B parameters from a pre-trained Qwen2.5-0.5B as the vertex models, and 0.6B randomly initialized parameters for the communication edges. Trained in an auto-regressive manner using a few standard public datasets, LMNet-1B achieves performance competitive with or even superior to existing open-source LLMs of similar size. Remarkably, this is accomplished with only 0.01T training tokens—less than 0.1% of the compute typically required to train a modern LLM from scratch (see Section 5.1). These results suggest a new path forward for scaling general intelligence: instead of increasing the size of a single model, we can compose smaller pre-trained LLMs using dense vector communication. We further show that this approach enables efficient and effective LLM customization with limited data, highlighting its potential for personalized and resource-constrained applications.

2 Related Works

We review existing works that build collective intelligence by constructing communication mechanisms between LLMs. Building collective intelligence with LLMs as modular components involves designing both the topology and functional roles of inter-model communication. These efforts can be broadly categorized into two types: multi-step reasoning and multi-model collaboration.

Multi-step reasoning [53, 4, 44], or more generally test-time scaling [30], aims to improve LLM performance by allocating additional computational resources during inference. Exemplar approaches include: Chain-of-Thought (CoT) Prompting [46]: Encourages LLMs to generate intermediate reasoning steps, improving accuracy on complex tasks at the cost of increased inference time; Parallel Sampling: like Majority Voting (self-consistency) [46, 39] and Best-of-N Sampling [48, 12], which generates multiple outputs at a single step and selects the best by certain strategies, though this can be computationally expensive; Process-Based Verifiers [36, 55]: which makes LLM generate intermediate reasoning steps with tree structure, and train models to evaluate intermediate steps (e.g., Process Reward Models or PRMs), enabling more efficient tree-search strategies. Note that considering the auto-regressive manner of text generation in LLM, even the naive CoT with a single model can be viewed as the model communicates with itself iteratively.

Multi-model collaboration [40, 25] builds complex workflows, including multi-agent systems powered by LLMs, to improve performance by leveraging specialized LLMs to divide complex tasks into subtasks. They can either be built by expert knowledge from human’s prior only, or data-driven. Without training data, exemplar works integrate standardized operating procedures into multi-agent workflows for software development [16], or design prompts to encourage LLM reasoning and acting with given tools iteratively [54] that. Given training data from the specific domain to for data-driven customization, existing works optimize by search [21, 59, 56, 58] or using LLM as black-box optimizer [26].

Despite their diversity, all of the above methods rely on natural language communication using discrete token sequences. This introduces inefficiencies in both information transfer and optimization. Two works partially move beyond this limitation. One work trains a single LLM to perform chain-of-thought reasoning in continuous space by modifying supervised fine-tuning to include unsupervised token expansion, but this approach does not generalize to complex reasoning structures or multi-agent collaboration and yields limited performance improvements [13]. Another one explores differentiable communication in reinforcement learning agents using a centralized neural controller, though it is not applicable to pre-trained LLMs or natural language tasks [37].

3 Method

Now we present the details of the proposed LMNet. We first remove the embedding and de-embedding layers of LLMs to enable dense vector communication. We then introduce the construction and training of LMNet.

3.1 Stripping Embedding Layers

Denote a tokenized text in natural language as $\vec{x}^{\text{in}} = [x_1, x_2, \dots, x_n]$, a sequence of discrete tokens where each token $x_i \in \mathcal{D}$ and $|\mathcal{D}|$ is the vocabulary size. The embedding layer E embeds the discrete tokens into dense vectors, $\mathbf{X}^{\text{in}} = E(\vec{x}^{\text{in}})$, where $\mathbf{X}^{\text{in}} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]$ and $\mathbf{x}_i \in \mathbb{R}^d$. Inside a LLM, the transformer model T takes \mathbf{X}^{in} as input, and output dense vectors with the same size, denoted as $\mathbf{X}^{\text{out}} = T(\mathbf{X}^{\text{in}})$. The de-embedding layer D decodes the dense vectors to discrete tokens to output natural language text, denoted as $\vec{x}^{\text{out}} = D(\mathbf{X}^{\text{out}})$. The complete function of a LLM f is $\vec{x}^{\text{out}} = D \circ T \circ E(\vec{x}^{\text{in}}) = f(\vec{x}^{\text{in}})$.

Given two LLMs f_1, f_2 to build a system with communication flow $f_1 \rightarrow f_2$, the existing and natural way is to let them communicate with natural language by setting $\vec{x}_2^{\text{in}} = \vec{x}_1^{\text{out}}$, i.e., $\vec{x}_2^{\text{out}} = D_2 \circ T_2 \circ E_2 \circ D_1 \circ T_1 \circ E_1(\vec{x}^{\text{in}}) = f_2 \circ f_1(\vec{x}^{\text{in}})$. We let them communicate with dense vectors by setting $\mathbf{X}_2^{\text{in}} = \mathbf{X}_1^{\text{out}}$, i.e., removing the internal de-embedding layer D_1 and embedding layer E_2 such that $\vec{x}_2^{\text{out}} = D_2 \circ T_2 \circ T_1 \circ E_1(\vec{x}^{\text{in}})$.

3.2 Constructing LMNet

While it is typical to formulate the hypothesis of workflow of LLM systems as a directed graph, where vertexes are LLMs and edges are communication paths, we specify it as a layer-wise fully connected feed-forward network for simplicity, akin to the structure of MLPs. Formally, denote LMNet as $\mathcal{N} = (\mathcal{V}, \mathcal{E})$ where:

- Each vertex $v \in \mathcal{V}$ represents T (a stripped transformer without embedding and de-embedding layer). The vertexes are arranged in L layers: $\mathcal{V} = \{\mathcal{V}^l\}_{l=1}^L = \{\{v_i^l\}_{i=1}^{n_l}\}_{l=1}^L$. Specially, there is only one vertex at the last layer as the output vertex, $n_L = 1$, whose de-embedding layer D_1^L is kept to convert dense vectors to discrete tokens in natural language for final output. And there is only one vertex at the first layer as the input vertex, $n_1 = 1$, whose embedding layer E_1^1 are kept to convert discrete tokens in initial input natural language to dense vectors. We denote $v_1^0 = E_1^1$ for notation consistency.
- Each edge $e \in \mathcal{E}$ represents a communication pathway, through a trainable seq2seq module parameterized by ω . As a layer-wise fully connected structure, there are and only are $\{e_{i,j}^l \mid \forall v_i^l \in \mathcal{V}^l, \forall v_j^{l+1} \in \mathcal{V}^{l+1}\}$.

Given initial input text \vec{x}^{in} , the kept embedding layer E_1^1 embed it to dense vectors \mathbf{X}^{in} . Then \mathcal{N} works in the way like a feed-forward neural network does. Formally, initializing $\mathbf{X}_1^0 = \mathbf{X}^{\text{in}}$, we have

$$\mathbf{X}_j^{l+1} = v_j^{l+1} \left(\sum_{v_i^l \in \mathcal{V}^l} e_{i,j}^l(\mathbf{X}_i^l; \omega_{ij}^l); \theta_j^{l+1} \right), \quad (1)$$

where θ_j^{l+1} is the parameter in vertex module v_j^{l+1} and ω_{ij}^l is the parameter in edge module $e_{i,j}^l$. The final output \mathbf{X}_1^L is de-embedded to text $\vec{x}^{\text{out}} = D_1^L(\mathbf{X}_1^L)$, and mapped to the output logits $p^o \in \mathbb{R}^{n \times |\mathcal{D}|}$ for optimization. In this way, such a LMNet takes natural language as input and output the same way as a LLM system typically does, thus can be applied for general NLP tasks.

3.3 Training

One of the most significant benefits of communication through dense vectors is that the path is differentiable, i.e., any gradients in $\partial \mathbf{X}_j^m / \partial \mathbf{X}_i^l$, $\partial \mathbf{X}_i^l / \partial \theta_i^l$ and $\partial \mathbf{X}_i^l / \partial \omega_{ki}^{l-1}$ are tractable. This leads to Proposition 1 which enables joint and efficient optimization by end-to-end gradient descent.

Proposition 1 (End-to-End Gradient Descent). *Given a differentiable supervision signal \mathcal{L} , i.e., $\frac{\partial \mathcal{L}}{\partial p^o}$, we can obtain gradient on all parameters in LMNet, i.e., $\frac{\partial \mathcal{L}}{\partial \theta_i^l}$ and $\frac{\partial \mathcal{L}}{\partial \omega_{ij}^l}$ for any l, i, j .*

Subsequently, we can obtain gradients of all vertex and edge parameters by the chain law:

$$\frac{\partial \mathcal{L}}{\partial \theta_i^l} = \frac{\partial \mathcal{L}}{\partial p^o} \cdot \frac{\partial p^o}{\partial \mathbf{X}_1^L} \cdot \frac{\partial \mathbf{X}_1^L}{\partial \mathbf{X}_i^l} \cdot \frac{\partial \mathbf{X}_i^l}{\partial \theta_i^l}, \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial \omega_{ij}^l} = \frac{\partial \mathcal{L}}{\partial p^o} \cdot \frac{\partial p^o}{\partial \mathbf{X}_1^L} \cdot \frac{\partial \mathbf{X}_1^L}{\partial \mathbf{X}_j^{l+1}} \cdot \frac{\partial \mathbf{X}_j^{l+1}}{\partial \omega_{ij}^l}.$$

Thus, all stages in the pipeline of training a single LLM can be applied to train LMNet, including large-scale pre-training, supervised fine-tuning, training by reinforcement learning. We recommend to treat θ and ω differently, as θ has already been pre-trained with fine-grained information, while ω is randomly initialized that contains no basic knowledge at all. It is recommended to first do pre-training with naive auto-regressive loss optimizing ω only to equip LMNet with basic ability to model natural language.

Specifically, the simplest case to train and use LMNet is customizing given LLM with limited data: the scenario given a pre-trained LLM and some text data for adaptation. In this case, we construct LMNet with the pre-trained LLM as vertex, and train by gradient descent. When the training data is few, it is recommended to optimize ω only, for parameter efficiency. The implementation of training process under different application scenarios would be presented in Section 5.

4 Discussion

4.1 Specification of Modules

For edge modules, any seq2seq modules are applicable. To be parameter-efficient yet expressive enough, we chose to use a single attention-block (1-layer transformer) for each edge. All edge modules would be independently and randomly initialized. For vertex models, LMNet does not require all the vertex models to be identical. We can implement each vertex with different pre-trained LLMs respectively, to exploit that different LLMs may have different expertise and LMNet can

combine them together. However, we can choose a much more parameter-efficient way: implementing all the vertex with a single pre-trained LLM. As different edges are different, the input information of different vertex would be different, so LMNet would still be effective. For the aggregation function of messages from multiple input edges of one vertex, we use sum for simplicity: in this case we can use the same causal mask for all vertexes and edges, to keep the causal structure of the input sequence of pre-trained LLMs.

4.2 Topology Hypothesis

We construct the layer-wise fully connected structure as the hypothesis of workflow of a LLM system by default. This is inspired by the similar topology among activation functions in MLP. Akin to the universal approximation property of MLPs, LMNet can express a wide range topology and functions of workflows. For example, one can easily figure out chain/tree/graph structures inside the fully-connected structure.

4.3 Complexity Analysis

Though the empirical results in Section 5.1 would include both computation resources and time consumption, here we provide a complexity analysis to better understand how LMNet works, and mention important implementations. Denote the parameters in LMNet as θ and ω as the collection of vertex and edge parameters respectively. Denote the depth of LMNet as L and average width as W , approximately with $L \times W^2$ edges following the layer-wise fully-connected topology. Denote the average time-consumption of feeding-forward as t_θ through a vertex and t_ω through an edge. With vertex parameter sharing, which means given a single pre-trained LLM θ_v , i.e., initializing all θ_i^l with θ_v and keeping $\theta_i^{l_1} = \theta_j^{l_2}$ for any l_1, l_2, i, j during training, the parameter size is $|\theta| + |\omega| = |\theta_v| + L \times W^2 \times |\omega|$ where $|\omega| \ll |\theta_v|$. This means when the size of LMNet grows, its parameter size only grows by a small ratio. This gives the possibility to build a deep and wide LMNet. For time complexity, keeping vertexes in the same layer identical enables simply parallelizing them through ‘`torch.DataParallel`’. In this case, a feeding-forward process only requires a sequential processing with length L , like a MLP, and $t = L \times t_\theta + L \times W^2 \times t_\omega$, where $t_\omega \ll t_\theta$.

4.4 Collective Intelligence in LMNet

By taking pre-trained LLMs as relative lower-level intelligence, LMNet embodies the collective intelligence, the important idea that has been widely claimed and practiced. In LMNet, each vertex (a stripped transformer from pre-trained LLM) acts as a modular component, while the edges (trainable seq2seq modules) facilitate differentiable communication, enabling the system to collectively solve problems beyond the reach of any single LLM. By optimizing these communications, it can be expected that LMNet not only scales computational power but also fosters emergent behaviors, such as advanced reasoning and adaptive problem-solving, that transcend the capabilities of its individual components. LMNet, is likely to be a more reasonable way than enlarging a single LLM.

5 Applications and Empirical Results

In this section, we discuss two exemplar applications of LMNet and provided empirical results. The first one is **scaling for general intelligence**. Like training a single LLM from scratch, we do our best in a data-rich way to build a LMNet that can understand human’s instructions for diverse tasks and generate coherent natural language. The second one is **customizing with limited data**. Given limited training set on a domain-specific problem, we build LMNet to adapt to the given training data. Code will be released to public.

5.1 Scaling for General Intelligence

The pursuit of general intelligence through LLMs has traditionally focused on scaling individual models [20, 5, 1, 11, 52, 42], either by increasing their parameter count or enhancing their training data. However, this approach faces diminishing returns due to computational costs of the challenges of further optimizing monolithic architectures, and the gradual consumption of high-quality data. Our

Table 1: Performance comparison of pre-trained LLMs (Accuracy, %).

Model	Qwen2.5-0.5B	LMNet-1B	Llama3.2-1B	Qwen2.5-1.5B	Gemma2-2B	Llama3.2-3B
# Parameters	0.49B	1.14B	1.23B	1.54B	2.61B	3.21B
# Training Tokens	18T	0.01T	15T	18T	15T	15T
General Tasks	MMLU	44.3	53.9	32.2	60.9	52.2
	MMLU-pro	15.7	26.2	12.0	28.5	23.0
	BBH	20.3	47.3	31.6	45.1	41.9
	ARC-C	35.6	38.0	32.8	54.7	55.7
	Truthfulqa	40.2	47.9	37.7	46.6	36.2
Math & Science	GSM8K	41.6	50.3	9.2	68.5	30.3
	MATH	19.5	38.8	-	35.0	18.3
	GPQA	24.8	25.6	7.6	24.2	25.3
	MMLU-stem	39.8	46.0	28.5	54.8	45.8
Coding	HumanEval	30.5	39.0	-	37.2	19.5
	MBPP	39.3	45.8	-	60.2	42.1

proposed LMNet paradigm offers a alternative by leveraging existing pre-trained LLMs as modular components within a densely connected network.

From a technical perspective, comparing with training a single LLM, LMNet has advantage in training cost and data requirement. Note that people need to train from scratch to build a larger LLM, and keep requiring new high-quality data to enhance performance. The first problem is training from scratch is wasting existing pre-trained LLMs which have already encoded vast information. LMNet addresses this by utilizing pre-trained LLMs in vertexes. The second problem is high-quality data is gradually depleted. LMNet addressed this by that LMNet can be trained with the data that has been used for training vertex LLMs, to enable the communication pathways and adapt transformer to dense vector inputs. Note that existing collection of LLMs could not be optimized for general intelligence effectively, because they communicate through natural language, disabling large-scale efficient optimization through gradient-descent.

5.1.1 Performance of (Re)Pre-Trained LMNet

Due to the computation budget, we consider modern LLMs with minimum size. We implement LMNet with 5 layers with 1/4/4/4/1 vertexes in each layer. We use Qwen2.5-0.5B as vertex module to shared among all vertexes. We use an attention block with the same structure as one transformer layer in the vertex module for each edge module. These edge modules are independently random initialized and to be optimized. Such a LMNet has 1.1B parameters in total. We use typical auto-regressive loss for training (from head to tail without distinguishing prompts and answers in each text). All data we used comes from public datasets C4 [34], Alpaca [41], ProsocialDialog [22], LaMini-instruction [51], MMLU [14] (auxiliary_training split only), MATH [15] (training split only), GSM8K [8] (training split only). Like advised in Section 3.3, we first freeze vertex parameters and update edge parameters, then we update all parameters together. Due to computation resource budget, **we train LMNet for less than 60 GPU-days (NVIDIA A100), and cost at most 0.01T tokens (only a very small subset of above mentioned datasets) or 1e5 PFLOPs in total**. In terms of either tokens or computation cost or time, the training cost of LMNet-1B is significantly less than the pre-training cost for comparable LLMs (less than 0.1%).

We compare with modern pre-trained-only LLMs in similar size, including Qwen-0.5B/1.5B [52], Llama3.2-1B/3B [11], Gemma2-2B [43], on widely-used benchmarks (with commonly-used benchmark-specific prompt) MMLU [14] (5-shot), MMLU-Pro [47] (5-shot, CoT), BBH [38] (3-shot, CoT), GSM8K [8] (4/8-shot, CoT), MATH [15] (0/4-shot, CoT), GPQA [35] (5-shot, CoT), HumanEval [6] (0-shot), MBPP [2] (0-shot). The performance is provided in Table 1. First, LMNet-1B brings comprehensive and significant improvement over the vertex model Qwen2.5-0.5B. This gives a way to improve general performance given a pre-trained LLM, using public data and acceptable training cost. Second, comparing LMNet-1B with other open-source pre-trained LLMs with similar

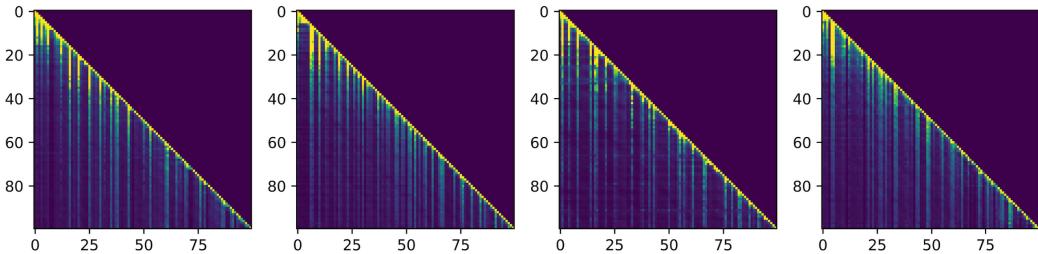


Figure 3: Visualization of attention weights in the edge modules on the 4 edges at the last layer of trained LMNet-1B, given certain input sentence (first 100 tokens only).

or slightly larger size, LMNet-1B shows comparable or even better performance, with less than 0.1% training cost. This gives an efficient and effective way to scale for general intelligence by utilizing existing pre-trained LLMs rather than train single LLM from scratch.

Note that we aim at providing a new method rather than the trained LMNet-1B ready-to-use. As we had very limited computation budget, we expect easily further improvement and more significant performance, by larger-scale training with more deliberate data and schedule, deeper and wider LMNet structure, larger and more diverse vertex modules, and integrating post-training and reinforcement learning.

5.1.2 A Closer Look at How LMNet Makes Inference through Dense Communication

We take a closer look at the trained LMNet-1B, to see what edge topology and functions have been learned and how the vertexes communicate to make inference as a collective intelligence.

First, we try to find some topology patterns across the edges. We visualize the parameters in all edge modules of the edges connecting the 1/4/4/4/1 vertexes, provided in Appendix A. We find that all edges are very different from initialization and each other, but no significant different statistical pattern between them can be observed. This indicates that the layer-wise fully connected structure is fully exploited through training, that it does not collapse to a simpler structure, e.g., chain or tree.

Then, we perform case study to see the inference process. We take the first test case in GSM8K, with input: "Question: Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for 2 per fresh duck egg. How much in dollars does she make every day at the farmers' market? Answer:". LMNet-1B answers correctly with "Janet has 16 eggs per day and eats 3 eggs per day, so she has $16 - 3 = 13$ eggs left.\nShe bakes muffins for her friends every day with 4 muffins per day, so she has $13 - 4 = 9$ muffins left.\nShe sells the remaining 9 muffins at the farmers' market for 2 per muffin, so she makes $9 * 2 = 18$ dollars every day.\n#### 18".

To see if the communication makes effect, we first try to de-embed the input dense vector sequences of different intermediate vertexes. But we failed with most token typically shows very close logits on many words, and even if we force to find a word with hard-max logit, the outcome input text is not even English. We assume this is because we implement the topology with layer-wise fully connected and the aggregation function of different input edges on a single vertex with simply summation, that fuses a lot; and the joint optimization of vertex, edges, embedding parameters. However, we do observe that the inputs of different vertexes are noticeably different. To show this, we visualize the attention weight in attention blocks of edge modules on the 4-edges at last layer (connecting the 4 vertexes at the second to last layer and the only output vertex at the last layer). As shown in Figure 3, we can find that there are obvious differences among the four edges, which indicates that different edges pass different messages.

5.2 Customizing LLM with Limited Data

Another scenario where training a LMNet is much more efficient and also can do great help, is adapting a pre-trained LLM with limited data. As the data is limited, we should not update too much parameters to avoid over-fitting. The importance of collective intelligence [29] and structure of LMNet gives an option: we can build a collective intelligence system and let the limited data determines the communications only, by constructing a LMNet and freezing the vertexes with weights from pre-trained LLMs, training few edge parameters only.

Table 2: Performance comparison on MMLU dataset (ΔAcc , %).

	GPT2-XL	Llama3.2-1B	Llama3.2-1B-Instruct	Qwen2.5-0.5B	Qwen2.5-1.5B
Pred	0.24	6.05	20.93	22.36	34.75
Prompt	1.69	11.69	24.30	22.50	35.83
FT	7.40	24.65	24.83	21.50	35.02
LMNet	13.10	27.19	27.57	23.35	37.28

Table 3: Performance comparison on GSM8K dataset (Accuracy, %).

	Llama3.2-1B	Llama3.2-1B-Instruct	Qwen2.5-0.5B	Qwen2.5-1.5B
Pred	2.88	30.10	5.31	9.25
Prompt	11.49	43.67	41.60	68.50
FT	25.32	35.63	24.11	51.08
LMNet	33.41	45.75	<u>30.93</u>	<u>60.02</u>

We consider fair comparison under a strict setting that only a pre-trained LLM and a training set are accessible, to improve the performance on unseen testing set from the same domain. Generally, we can consider three categories of methods: (i) prompting, including demonstrating examples from training set for ICL and prompting to reasoning better like CoT ; (ii) fine-tuning, including parameter-efficient fine-tuning methods; (iii) training LMNet. We consider training and testing on benchmark datasets MMLU, GSM8K and E2E [31] respectively.

5.2.1 Performance Comparison on MMLU and GSM8K

We study on widely used benchmark MMLU and GSM8K to show the effect of LMNet on different models. The baselines include **Pred**: directly ask the LLM to answer the question; **Prompt**: use model- and dataset- specific prompt engineering to optimize the performance (typically and at least combines ICL and CoT); **FT**: fine-tune the LLM with the training set; **LMNet**: construct LMNet with the given LLM as vertex and train with training set.

For both datasets, we construct LMNet with a small scale (3 layers with 1/2/1 vertexes), and keep all the vertexes share the same group of parameters for efficiency. All parameters in LMNet are updated together by gradient descent. We use a module with the same structure with a single transformer layer from the corresponding backbone LLM. Taking Qwen2.5-1.5B for example, a single LLM model has 1.76×10^9 parameters, while constructing the LMNet only introduce additional 2.45×10^8 parameters on the edge modules. In this case, the parameter size of LMNet is close to a single model, and the cost of training LMNet and fine-tuning a single model have similar scale.

The performance on MMLU is evaluated by ΔAcc , which is the difference between testing accuracy and random guess average accuracy (25%), provided in Table 2. LMNet show significant performance advantage over any other adaptation methods on every considered LLM.

The performance on GSM8K is provided in Table 3. Given Llama3.2-1B or Llama3.2-1B-Instruction as backbone LLM, LMNet still performs the best. However, given Qwen2.5-0.5B or Qwen2.5-1.5b, LMNet fails in comparison with Prompt, and the performance of Prompt and LMNet on Llama3.2-1B-Instruction are close. We infer the following two factors together cause such result. First, these three LLMs are strong enough for such tasks with a single model given proper instructions, that the additional communication steps brought by LMNet do little help. This can also be witnessed in Table 2, from the relative close performance between different methods on these three LLMs comparing with the other weak LLMs. Second, the training set of GSM8K is much smaller than MMLU (7.47k sentences vs 100k sentences). This result in overfitting risk for methods that updates model parameter, including FT and LMNet. Despite of this factor, comparing with FT, LMNet still shows advantage. Second, these three LLMs are strong enough for such tasks with a single model given proper instructions, that the additional communication steps brought by LMNet does little help.

5.2.2 Parameter-Efficient Fine-tuning on E2E Dataset

The results on GSM8K expose the parameter-efficient issue of adapting LLM with limited data. So in this part, we compare variants of LMNet with different structures and training strategies, with other parameter-efficient fine-tuning (PEFT) methods. Following the experiment setting in LoRA[18],

Table 4: Performance comparison on E2E dataset with GPT2-M. * indicates results from LoRA.

Method	# Training Parameters	Metrics				
		BLEU	NIST	MET	ROUGE-L	CIDEr
-	0	0.00	0.42	0.04	0.16	0.00
FT*	354.92M	68.2	8.62	46.2	71.0	2.47
Adapter ^L * [24]	0.37M	66.3	8.41	45.0	69.8	2.40
Adapter ^L * [24]	11.09M	68.9	8.71	46.1	71.3	2.47
Adapter ^H * [17]	11.09M	67.3	8.50	46.0	70.7	2.44
FT ^{Top2} * [23]	25.19M	68.1	8.59	46.0	70.8	2.41
PreLayer [*] [23]	0.35M	69.7	8.81	46.1	71.4	2.49
LoRA [18]	0.35M	68.9	8.68	46.5	71.5	2.51
LMNet-1/1	21.00M	69.1	8.76	46.6	70.6	2.43
LMNet-1/2/1	36.75M	70.5	<u>8.85</u>	45.7	71.5	2.38
LMNet-1/2/2/1	57.75M	69.8	8.80	46.0	71.1	2.49
LMNet-1/4/1	57.75M	69.1	8.70	45.9	70.9	2.44
LMNet-1/4/4/1	141.75M	68.7	8.79	46.5	71.2	2.48
LMNet-1/2/1 + Prefix	36.75M	70.5	8.88	46.2	72.4	2.46
LMNet-1/2/1 + FT	391.67M	66.0	8.46	42.4	68.3	2.05
LMNet-1/2/1 + LoRA	37.10M	70.1	8.82	46.2	71.7	2.54

we study with GPT2-M [33] model on E2E dataset, and consider the same group of PEFT methods [17, 23, 18].

To achieve parameter-efficient with LMNet, by default we only train the edge parameters and keep the vertex parameters frozen. We use an attention block, containing 5.25M parameters, for each edge module. The results are provided in Table 4, where we use LMNet- $n_1/n_2/\dots/n_L$ to denote a LMNet with L layers with n_l vertexes on the l -th layer. Comparing among the second group, we find 1/2/1 to be a good size for this problem, that performs better than any traditional PEFT methods. Note that thanks to the fully-differentiable paths in LMNet, we can also integrate LMNet along with other PEFT methods. For example, LMNet+Prefix [23] means we add random-initialized prefix before the initial input \mathbf{X}^{in} . LMNet+FT means not only updating edge parameters, but also fine-tuning the pre-trained transformer, i.e., updating edge and vertex parameters together. LMNet+LoRA means not only training edge parameters, but also perform LoRA on the shared pre-trained transformer in vertexes. Plugging-in appropriate PEFT methods can further improve the performance of LMNet for customizing LLM with limited data.

6 Conclusion and Future Works

In this paper, we introduce LMNet, a novel paradigm for constructing collective intelligence by enabling direct dense vector communication between LLMs. By stripping the embedding and de-embedding layers, LMNet facilitates more efficient information transfer, fully differentiable optimization pathways, and the exploration of capabilities beyond human heuristics. For instances, we train LMNet-1B that achieves performance comparable to similar and even larger sized single LLMs while requiring less than 0.1% of the training cost, offering a scalable and cost-effective alternative to traditional LLM training. Additionally, we show empirical results that LMNet shows promise in customizing LLMs with limited data, highlighting its versatility.

This paper only provides the basic idea and primary practice, and there are many directions for further exploration. For example, improvement could be made by more deliberate choice of diverse LLMs as vertex modules, design of edge modules and aggregation functions. There are also many other reasonable topology choices, like several paths with no intermediate intersection, skip or residual connection, or mimicking message passing on given graph. One can define design the topology according to problem-specific prior knowledge, expertise in vertexes, and expressiveness theory. LMNet can also be equipped with multiple input/output vertexes to interact with environment to build LLM-based agent systems. We also hope future work to explore larger-scale collection and training, with refined training protocols, to release a well-trained large LMNet ready-to-use. Looking ahead, we hope that this work can inspire to open new avenues for scaling general intelligence by leveraging existing pre-trained models and optimizing their dense communications, which is more effective, efficient and eco-friendly than training single LLMs from scratch.

References

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.
- [3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3(Feb):1137–1155, 2003.
- [4] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of Thoughts: Solving elaborate problems with large language models. In *AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690, 2024.
- [5] 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. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, 2020.
- [6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.
- [7] KR1442 Chowdhary and KR Chowdhary. Natural language processing. *Fundamentals of Artificial Intelligence*, pages 603–649, 2020.
- [8] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- [9] Franciscus Cornelis Donders. On the speed of mental processes. *Acta psychologica*, 30:412–431, 1969.
- [10] Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *International Conference on Machine Learning*, 2023.
- [11] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- [12] Lin Gui, Cristina Gârbacea, and Victor Veitch. Bonbon alignment for large language models and the sweetness of best-of-n sampling. *arXiv preprint arXiv:2406.00832*, 2024.
- [13] Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.
- [14] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

- [15] 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. In *Advances in Neural Information Processing Systems*, 2021.
- [16] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. MetaGPT: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 3(4):6, 2023.
- [17] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.
- [18] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.
- [19] Daniel Kahneman. *Thinking, fast and slow*. macmillan, 2011.
- [20] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- [21] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, Heather Miller, et al. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *International Conference on Learning Representations*, 2024.
- [22] Hyunwoo Kim, Youngjae Yu, Liwei Jiang, Ximing Lu, Daniel Khashabi, Gunhee Kim, Yejin Choi, and Maarten Sap. ProsocialDialog: A prosocial backbone for conversational agents. In *Conference on Empirical Methods in Natural Language Processing*, 2022.
- [23] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.
- [24] Zhaojiang Lin, Andrea Madotto, and Pascale Fung. Exploring versatile generative language model via parameter-efficient transfer learning. *arXiv preprint arXiv:2004.03829*, 2020.
- [25] Guangyi Liu, Yongqi Zhang, Yong Li, and Quanming Yao. Dual reasoning: A GNN-LLM collaborative framework for knowledge graph question answering. In *Conference on Parsimony and Learning*, 2025.
- [26] Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic LLM-powered agent network for task-oriented agent collaboration. In *Conference on Language Modeling*, 2024.
- [27] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26, 2013.
- [28] George A Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63(2):81, 1956.
- [29] Marvin Minsky. *Society of mind*. Simon and Schuster, 1986.
- [30] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsumi Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.
- [31] Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. The E2E dataset: New challenges for end-to-end generation. *arXiv preprint arXiv:1706.09254*, 2017.
- [32] Steven Pinker. *The language instinct: How the mind creates language*. Penguin uK, 2003.
- [33] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

- [34] 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.
- [35] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. GPQA: A graduate-level google-proof Q&A benchmark. In *Conference on Language Modeling*, 2024.
- [36] Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*, 2024.
- [37] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, volume 29, 2016.
- [38] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.
- [39] Shuhei Suzuki and Keiko Hatano. Reducing hallucinations in large language models: A consensus voting approach using mixture of experts, 2024.
- [40] Yashar Talebirad and Amirhossein Nadiri. Multi-agent collaboration: Harnessing the power of intelligent llm agents. *arXiv preprint arXiv:2306.03314*, 2023.
- [41] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford Alpaca: An instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca, 2023.
- [42] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.
- [43] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussonot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.
- [44] Ye Tian, Baolin Peng, Linfeng Song, Lifeng Jin, Dian Yu, Lei Han, Haitao Mi, and Dong Yu. Toward self-improvement of llms via imagination, searching, and criticizing. In *Advances in Neural Information Processing Systems*, volume 37, pages 52723–52748, 2024.
- [45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.
- [46] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.
- [47] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyuan Jiang, et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2024.
- [48] Zihao Wang, Chirag Nagpal, Jonathan Berant, Jacob Eisenstein, Alex D’Amour, Sanmi Koyejo, and Victor Veitch. Transforming and combining rewards for aligning large language models. *arXiv preprint arXiv:2402.00742*, 2024.
- [49] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837, 2022.

- [50] Terry Winograd. Understanding natural language. *Cognitive Psychology*, 3(1):1–191, 1972.
- [51] Minghao Wu, Abdul Waheed, Chiyu Zhang, Muhammad Abdul-Mageed, and Alham Fikri Aji. LaMini-LM: A diverse herd of distilled models from large-scale instructions. In *Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 944–964, 2024.
- [52] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.
- [53] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822, 2023.
- [54] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.
- [55] Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. ReST-MCTS*: LLM self-training via process reward guided tree search. In *Advances in Neural Information Processing Systems*, volume 37, pages 64735–64772, 2024.
- [56] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. AFlow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024.
- [57] Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Zhihan Guo, Yufei Wang, Irwin King, Xue Liu, and Chen Ma. What, how, where, and how well? a survey on test-time scaling in large language models. *arXiv preprint arXiv:2503.24235*, 2025.
- [58] Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulić, Anna Korhonen, and Sercan Ö Arič. Multi-agent design: Optimizing agents with better prompts and topologies. *arXiv preprint arXiv:2502.02533*, 2025.
- [59] Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbulin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *International Conference on Machine Learning*, 2024.

A Visualization of Edges

We visualize query/key/value/output projection matrix of the attention block on every edge in trained LMNet-1B respectively. Here we only provide the results of query projection matrix in Figure 4, as the others show very similar pattern.

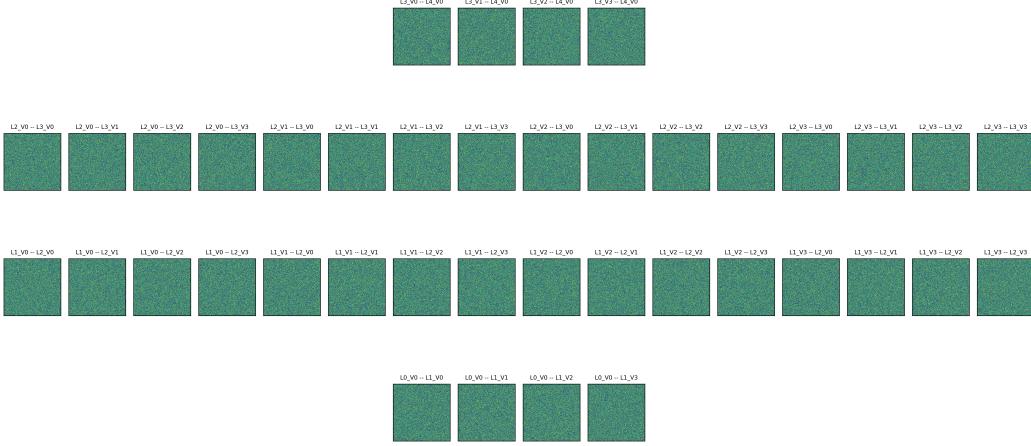


Figure 4: Visualization of query projection matrix of the attention block on every edge in trained LMNet-1B. All edges are shown under the same value-color mapping.