GOOD AGENTIC FRIENDS DO NOT JUST GIVE VERBAL ADVICE: THEY CAN UPDATE YOUR WEIGHTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent LLM systems usually collaborate by exchanging natural-language messages. This interface is simple and interpretable, but it forces each sender's intermediate computation to be serialized into tokens and then reprocessed by the receiver, thereby increasing the generated-token cost, prefill overhead, and KV-cache memory. We study an alternative communication interface: instead of appending a sender's message to the receiver's context, compile the sender's hidden states into a transient, receiver-specific weight perturbation. We introduce **TFLOW** (Thought Flow), a weight-space communication framework for a known and fixed receiver architecture. For each query, frozen role-prompted sender agents process the input, and a learned parameter generator maps their internal activations into low-rank LoRA perturbations targeting the receiver's modules. These perturbations are fused and applied only during the receiver's generation, enabling instance-level adaptation without permanently changing the model or enlarging the receiver's text context. With three Qwen3-4B agents, **TFLOW** improves over a standalone receiver by up to 8.5 accuracy points across five benchmarks while reducing processed tokens by up to 32.69%. Compared with a text-based three-agent baseline, it reduces total processed tokens by up to 83.27% and the wall-clock inference time by up to 4.6×, while maintaining competitive accuracy on four of five benchmarks. These results suggest that transient low-rank weight perturbations can serve as an executable communication medium for efficient multi-agent LLM collaboration.

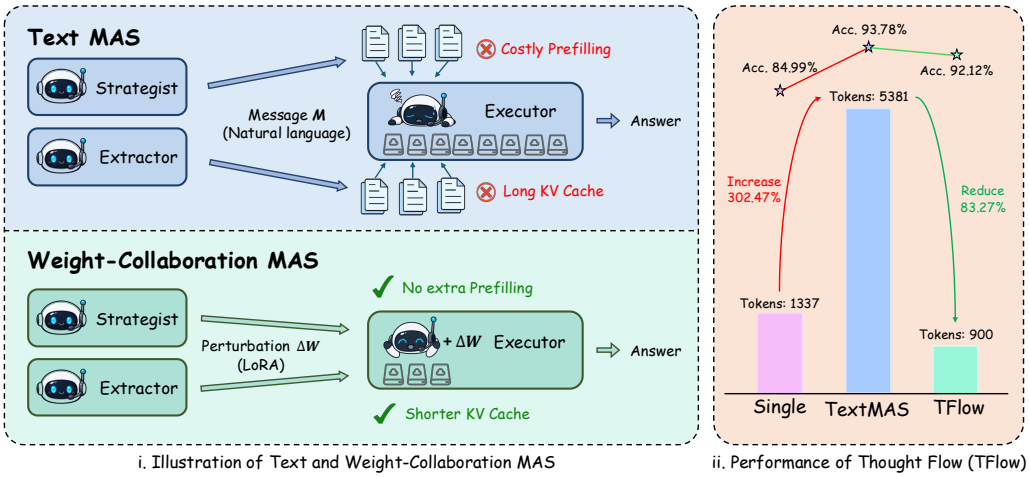
1 INTRODUCTION

Multi-agent LLM systems are often built like small committees. One model proposes a strategy, another contributes domain knowledge, and a final model produces the answer. This design has been effective across reasoning, coding, and tool-use settings [Li et al., 2023a, Hong et al., 2023, Wu et al., 2024, Du et al., 2024, Liang et al., 2024, Guo et al., 2024, Xi et al., 2025]. Yet the communication interface between agents has changed surprisingly little: most systems still require each agent to write a natural-language message that is appended to another agent's context.

This text interface is attractive because it is universal and human-readable. **Yet we argue that it may be an awkward interface between neural networks.** A sender agent has already transformed the input into a rich internal state, but conventional multi-agent systems force this state to be decoded into tokens before the receiver can use it. The receiver must then encode those tokens again, paying additional prefill cost and storing a larger KV cache (Figure 1 (i)), whose memory and latency grow with context length [Pope et al., 2023, Kwon et al., 2023]. As the number of agents or communication rounds grows, this repeated write-read cycle can dominate the cost of collaboration. The issue is not only that natural-language messages are long; it is that they make model-to-model communication pass through a channel designed for humans.

Recent work has begun to question this design by allowing agents to exchange continuous representations rather than text [Pham et al., 2024, Du et al., 2025, Zheng et al., 2025, Zou et al., 2025, Jin et al., 2026]. These latent-space protocols reduce decoding overhead and can preserve information that would be difficult to express in words. However, they still require the receiver to consume sender information as activations, embeddings, or cache states. This creates a compatibility problem: the transmitted object must be meaningful inside the receiver's representation geometry, making such methods most natural when agents share architectures, hidden spaces, or trained adapters.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



i. Illustration of Text and Weight-Collaboration MAS ii. Performance of Thought Flow (TFlow)

Figure 1: **(i)** Text-based MAS vs. the proposed Weight-Collaboration MAS. In Text MAS, auxiliary agents send natural-language messages to the Executor, incurring costly prefilling and an inflated KV cache. Our paradigm instead compresses inter-agent communication into lightweight LoRA perturbations ΔW merged directly into the parameters, eliminating the extra prefilling and shrinking the KV cache. **(ii)** Performance on GSM8K: TFLOW matches TextMAS in accuracy with $\mathbf{83.27}\%$ fewer tokens, and surpasses the single-agent baseline in both accuracy and efficiency.

We explore a complementary route. *Instead of sending information that the receiver must read, we send information that changes how the receiver computes.* In particular, we ask whether a sender's internal state can be compiled into a temporary low-rank perturbation of the receiver's weights [Houlsby et al., 2019, Hu et al., 2022]. The receiver then solves the query under this perturbed computation, after which the perturbation is removed, and the frozen base model is restored. In this view, the communication object is not a sentence and not a hidden state injected into the receiver's cache. It is a query-specific update in the receiver's parameter space.

We instantiate this idea in **TFLOW** (Thought Flow), a framework for weight-space communication among LLM agents. We focus on a practical fixed-receiver setting: the receiver architecture and target modules are known, all LLM backbones remain frozen, and only a receiver-specific parameter generator [Mahabadi et al., 2021, Ha et al., 2016] is trained. Given a query, each sender agent processes the prompted input once and exposes its hidden states as a conditioning signal. The parameter generator maps this signal into layer- and module-specific LoRA factors for the receiver. Perturbations from multiple senders are fused and injected by transiently patching the forward pass of the receiver's target linear layers. The receiver then generates its answer without seeing any sender-written message in its text context (Figure 1 (ii)).

This design separates TFLOW from both text-based and latent-space collaboration. Compared with text-based MAS, TFLOW avoids expanding the receiver's input with auxiliary messages, thereby reducing generated-token usage, prefill overhead, and KV-cache growth. Compared with latent-state transfer, TFLOW does not ask the receiver to directly interpret a sender's activations. Sender states are instead translated by a learned generator into perturbations that are compatible with the receiver's own weight tensors by construction. The result is an executable communication channel: sender information affects the receiver by modifying the computation path used for the current query.

We evaluate TFLOW in a three-agent setting with a frozen Qwen3-4B backbone, where sender agents provide different perspectives through hidden states and the receiver produces the final answer. Across five benchmarks, TFLOW improves over the standalone receiver by up to 8.5 accuracy points while reducing total processed tokens by up to 32.69%. Relative to a text-based three-agent baseline, it cuts processed tokens by up to 83.27% and wall-clock inference time by up to 4.6x, with competitive accuracy on four of five benchmarks; the remaining gap on HumanEval+ suggests that additional generation budget matters more than inter-agent communication for code synthesis.

Our contributions are:

- Motivated by the high communication cost of text-based multi-agent systems, we propose ***weight-space communication***, a new communication paradigm for multi-agent LLM collaboration, in

which sender information is transmitted as transient low-rank perturbations to a frozen receiver model rather than as text appended to the receiver's context.

- Under this paradigm, we introduce **TFLOW**, a receiver-specific framework that transforms sender hidden states into query-specific LoRA factors, integrates multiple sender contributions, and injects the resulting perturbation only during the receiver's generation.
- We demonstrate that TFLOW achieves substantial efficiency gains across five reasoning, knowledge, and coding benchmarks, substantially reducing token and latency costs relative to text-based multi-agent collaboration while remaining competitive in accuracy on four of five benchmarks, and consistently outperforming a standalone receiver.

2 Related Work

2.1 LLM-Based Multi-Agent Systems

Multi-Agent Collaboration Paradigms. Large language models have catalyzed a wave of multi-agent systems. Early explorations such as CAMEL [Li et al., 2023a] and Generative Agents [Park et al., 2023] demonstrate the power of role-playing agents that interact through natural-language conversations. Subsequent work applies multi-agent collaboration to complex engineering workflows: MetaGPT [Hong et al., 2023] and ChatDev [Qian et al., 2024] mirror real software companies by assigning specialized roles and routing structured artifacts between them, while AutoGen [Wu et al., 2024] and AutoAgents [Chen et al., 2023] provide general-purpose frameworks for orchestrating and automatically generating task-specific agents. Other work leverages multi-agent debate to improve reasoning quality [Du et al., 2024, Liang et al., 2024]. ReConcile [Chen et al., 2024] organizes round-table conferences among diverse LLM families with confidence-weighted voting. More recently, Mixture-of-Agents [Wang et al., 2025a] and Li et al. [2024] show that iteratively aggregating diverse models or simply scaling the number of sampled agents can surpass individual models. Despite their architectural diversity, all of the above systems share a common reliance on natural language as the primary medium for inter-agent communication.

Inter-Agent Communication Mechanisms. The communication mechanism is central to the effectiveness of any multi-agent system. The prevailing paradigm transmits information as natural-language messages concatenated into each receiver's context window [Li et al., 2023a, Hong et al., 2023, Wu et al., 2024, Qian et al., 2024]. While natural language provides a universal and human-interpretable interface, serializing an agent's intermediate computation into tokens may lose information that is difficult to verbalize and can introduce substantial communication overhead [Pham et al., 2024, Zheng et al., 2025, Du et al., 2024, Smit et al., 2024]. Recent work has therefore explored non-textual communication channels: CIPHER [Pham et al., 2024] lets agents communicate through probability-weighted output embeddings; Interlat [Du et al., 2025] transmits hidden states as a compressed representation of an agent's internal computation; and [Zheng et al., 2025, Tang et al., 2025, Zou et al., 2025] further formalize and extend latent-space communication. These approaches show that moving beyond natural-language messages can preserve richer intermediate signals and reduce token-level overhead, primarily by communicating in activation or embedding space. In this work, we explore a complementary paradigm: *weight-space communication*, where sender-side computation is converted into targeted, instance-specific perturbations of the receiver's model parameters, which guide the receiver's generation.

2.2 Knowledge Representation and Operations in Weight Space

Weight-Space Structure and Static Merging. A growing body of evidence shows that neural network weight spaces are structured carriers of transferable knowledge [Yang et al., 2026, Li et al., 2023b, Han et al., 2026]. The *task vector* framework [Ilharco et al., 2022] demonstrates that the element-wise difference between a fine-tuned model and its pretrained initialization encodes the learned task skill. Building on this insight, a family of model merging methods, including Model Soups [Wortsman et al., 2022], Fisher merging [Matena & Raffel, 2022], TIES-Merging [Yadav et al., 2023], and DARE [Yu et al., 2024], seeks to combine the knowledge of multiple fine-tuned models directly in weight space without additional training. Despite these advances, all of the above approaches operate in a *static* manner, in which the merging coefficients or perturbation directions are fixed once determined and do not adapt to individual inputs at inference time.

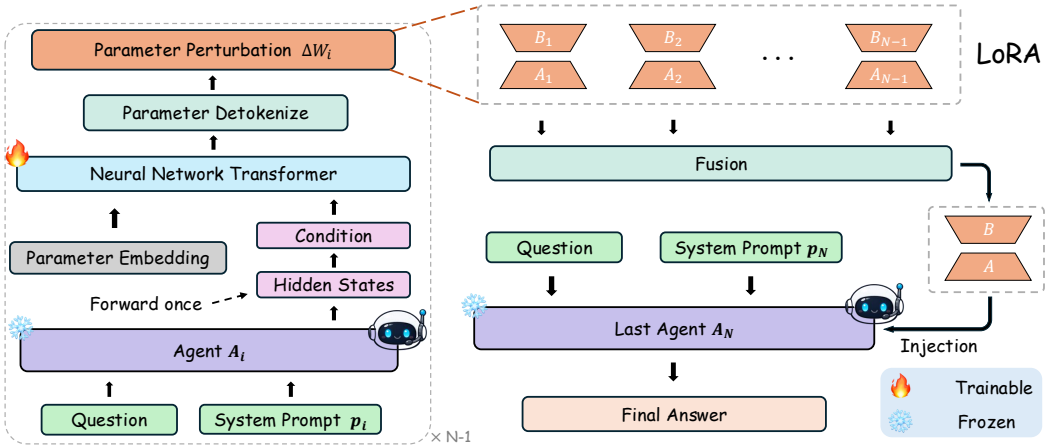


Figure 2: **Overview of TFLOW.** Each sender agent A_i runs one frozen forward pass on the question and its system prompt p_i; its hidden states condition a trainable parameter generator that emits LoRA factors $(\mathbf{A}_i, \mathbf{B}_i)$, i.e., a low-rank perturbation $\Delta\mathbf{W}_i$. The $N-1$ perturbations are fused into a single update $\Delta\mathbf{W}$ and transiently injected into the frozen receiver A_N, which generates the final answer under its own prompt p_N; the update is discarded afterwards, leaving the backbone unchanged. All agents remain **frozen**; only the parameter generator is **trainable**.

Low-Rank Adaptation and Dynamic Weight Generation. LoRAHub [Huang et al., 2024] consists of multiple LoRA modules with learned scalar weights for cross-task generalization, while LoRAs composition [Wang et al., 2024c] and LoRA-Flow [Wang et al., 2024a] enable dynamic adapter fusion at the instance level and the layer and token level, respectively. Going further, recent approaches use hypernetworks [Ha et al., 2016] to directly *generate* adapter parameters conditioned on input context—from task embeddings [Mahabadi et al., 2021], to few-shot demonstrations [Phang et al., 2023], to full documents [Charakorn et al., 2026, Liu et al., 2026]; others learn generators from trained checkpoint collections [Liang et al., 2025, Jin et al., 2024, Wang et al., 2024b; 2025b].

Our framework, TFLOW, extends this progression. Rather than conditioning on static documents or task embeddings, TFLOW uses a parameter generator to translate the *dynamic reasoning states* of multiple sender agents into low-rank perturbations of a receiver model's weights.

3 TFLOW: THOUGHT FLOW

Figure 2 illustrates the overall pipeline of TFLOW. This section formalizes the setup (Section 3.1), describes how sender hidden states are aggregated into query-dependent conditioning signals (Section 3.2), introduces the parameter generator that emits LoRA factors (Section 3.3), and details the transient injection into the receiver (Section 3.4).

3.1 PROBLEM SETUP OF WEIGHT-COLLABORATION MAS

We consider a collaboration of N agents $\{A_1, \ldots, A_N\}$ that share a single frozen backbone $f_{\boldsymbol{\theta}}$ but are individuated by role-specific system prompts $\{p_1, \ldots, p_N\}$. Given a query q, agents $A_1, \ldots, A_{N-1}$ act as *senders* and A_N as the *receiver* [Li et al., 2023a, Wu et al., 2024, Hong et al., 2023]. The senders contribute only their hidden-state trajectories, while the receiver is the one who actually decodes the answer. Rather than exchanging tokens, the senders induce, for this single query, a transient parameter displacement applied to a designated set of linear modules inside the receiver:

$$\hat{\boldsymbol{\theta}} \;=\; \boldsymbol{\theta} \;+\; \Delta\mathbf{W}\big(q, \{p_i\}_{i=1}^{N-1}; \boldsymbol{\psi}\big). \tag{1}$$

Here $\boldsymbol{\psi}$ denotes the parameters of a learned generator and is the *only* set of weights updated during training. The backbone tensors themselves are never overwritten: the perturbation is implemented as a temporary forward patch and removed immediately after the receiver call, so different queries always observe the same original backbone.

Two design constraints follow directly and will shape every component below. First, because we rely on the existence of beneficial neighbors rather than on aggressive fine-tuning, $\Delta\mathbf{W}$ must remain a

low-rank, small-norm displacement. Second, because useful perturbation directions are *instance-specific*, the generator must be conditioned on representations that depend on q itself, not merely on the static role descriptors p_i. We therefore condition on the senders' hidden states over the prompted input, which simultaneously encodes the role p_i and the content of the query.

Notation. In the remainder of this section, we restrict weight-space perturbations to L selected decoder layers, with M linear modules perturbed in each selected layer. For the m-th perturbed module in the l-th selected layer, we denote its frozen weight matrix by $\mathbf{W}_m^{(l)}$.

3.2 Sender Conditioning

Each sender A_i encodes the prompted input (p_i, q) using the frozen backbone in a single forward pass. We retain the intermediate activations and denote the resulting hidden states at layer l by $\mathbf{H}_i^{(l)} \in \mathbb{R}^{T_i \times d}$, for $l = 0, 1, \ldots, L_{\text{total}}$.

Learnable layer aggregation. Instead of relying on a single-layer representation, we aggregate each sender's hidden states across layers using temperature-scaled softmax weights over learnable layer-wise scalars. The resulting conditioning representation $\mathbf{C}_i \in \mathbb{R}^{T_i \times d}$ enables the generator to exploit features at different levels of abstraction while preserving a fixed conditioning dimensionality.

3.3 Parameter Generator

The parameter generator $\mathcal{H}_\psi$ maps each sender conditioning representation $\mathbf{C}_i$ to a complete set of LoRA factors spanning all $L \times M$ targeted modules in the receiver. The generator comprises three stages: conditioning-driven initialization, a multi-axis Transformer trunk, and detokenization into LoRA matrices.

Stage 1: Initialization. We construct a token grid of shape $(L, H \times W, d_{\text{pg}})$, where $H = H_A + H_B$ divides tokens into $\mathbf{A}$- and $\mathbf{B}$-generating groups, and W spans the target modules and rank slots. Learnable grid queries $\mathbf{Q}_{\text{grid}} \in \mathbb{R}^{L \times HW \times d_{\text{pg}}}$ are initialized by cross-attending to the projected conditioning representation $\text{Proj}_{\text{in}}(\mathbf{C}_i)$. We then add layer-index and module-rank positional embeddings to encode each token's structural location within the receiver.

Stage 2: Neural Network Transformer. Our generator architecture follows the structured parameter tokenization paradigm introduced by HY-WU [Team et al., 2026]. The initialized grid is refined through N_{pg} stacked blocks, each applying three residual attention operations along complementary axes, followed by a feed-forward network. Cross-layer self-attention (SA_L), batched over the HW axis, captures dependencies across receiver layers; intra-layer self-attention (SA_{HW}), batched over the L axis, propagates information across modules and rank slots within each layer; and conditioning cross-attention (CA) re-anchors the generation process to the sender representation. All attention operations employ RoPE [Su et al., 2024] along their active sequence axis.

Stage 3: Detokenization. After N_{pg} blocks, the refined grid is partitioned along the H dimension into $\mathbf{A}$- and $\mathbf{B}$-tokens. Two linear heads map these tokens to parameter slices, which are rearranged into per-sender LoRA factors $\mathbf{A}_{i,m}^{(l)} \in \mathbb{R}^{r \times d_{\text{in}}}$ and $\mathbf{B}_{i,m}^{(l)} \in \mathbb{R}^{d_{\text{out}} \times r}$. The generator is shared across all senders; sender-specific factors are induced solely by the conditioning representation $\mathbf{C}_i$.

3.4 Transient Injection

The generated factors are injected into each targeted module of the receiver as a low-rank weight offset. When multiple senders are active, their generated LoRA updates are fused by a lightweight scalar gate before being applied to the receiver:

$$\Delta \mathbf{W}_m^{(l)} = \frac{\alpha}{r} \sum_{i=1}^{N-1} \text{softmax}_i\big(g_{\mathbf{w}}(\mathbf{C}_i)\big) \mathbf{B}_{i,m}^{(l)} \mathbf{A}_{i,m}^{(l)}. \tag{2}$$

The resulting $\Delta \mathbf{W}_m^{(l)}$ is injected only into the designated receiver module m at layer l. During the receiver's forward pass, the frozen linear map is evaluated with this transient additive update. After

generation, the patch is discarded, ensuring that each input induces its own temporary parameterization and that all subsequent inputs start from the same frozen backbone.

3.5 Training Objective

Diversity regulariser. To prevent the generator from collapsing toward a query-independent LoRA, we penalise the squared cosine similarity $\mathcal{L}_{\text{div}}(\Theta; x_q) = \cos^2\big(v(x_q),\, v_{\text{prev}}^{(\sigma(x_q))}\big)$ between the flattened LoRA vector $v(x_q)$ of the current sample and the cached vector $v_{\text{prev}}^{(\sigma)}$ from the previous step. This loss is minimised at orthogonality, equally penalises parallel and anti-parallel collapse, and incurs no additional forward cost since the cache is updated via stop-gradient after each backward pass.

End-to-end optimization. The task loss $\mathcal{L}_{\text{task}}$ is the standard masked next-token cross-entropy computed only over completion tokens, with the receiver conditioned on both the chat-templated prompt and the injected ΔW. The full training objective combines it with the diversity regulariser:

$$\mathcal{L}(\Theta) \;=\; \mathbb{E}_{(x_q,\, t,\, \sigma)\sim\mathcal{D}}\Big[\mathcal{L}_{\text{task}}(\Theta; x_q, t) \;+\; \lambda_{\text{div}}\, \mathcal{L}_{\text{div}}(\Theta; x_q)\Big]. \tag{3}$$

The trainable parameter set Θ encompasses all components of the parameter generator $\mathcal{H}_{\psi}$, the conditioning projection layers, and the gating head; the receiver and sender backbones share the frozen parameters $\boldsymbol{\theta}$ and remain entirely frozen throughout training. We do not add auxiliary weight-norm or inter-sender cosine penalties in our default objective; empirically, the cache-based diversity regularizer is sufficient to encourage instance-level specialization.

4 Experiments

4.1 Experimental Setups

Datasets and evaluation. We evaluate on five benchmarks spanning mathematical reasoning, code generation, and multi-disciplinary knowledge: GSM8K [Cobbe et al., 2021], MATH [Hendrycks et al., 2021], MMLU-Redux [Gema et al., 2025, Hendrycks et al., 2020], HumanEval+ [Liu et al., 2023, Chen et al., 2021], and MBPP+ [Liu et al., 2023, Austin et al., 2021]. All benchmarks are evaluated under the zero-shot, chain-of-thought setting with no in-context exemplars. At inference time, all methods use identical decoding hyperparameters, which is temperature 0.6, top-p 0.95.

Backbone and baselines. All methods share a single frozen **Qwen3-4B** [Yang et al., 2025] backbone, whose parameters are never updated during parameter generator training or inference. We compare against two baselines: (i) **Single-Agent**, which directly prompts Qwen3-4B with a task-level system prompt and represents the backbone's intrinsic capability; (ii) **TextMAS**, a three-agent text-level collaboration in which Agents A and B produce strategy and domain-knowledge passages that are concatenated into Agent C's context, representing conventional multi-agent cooperation.

Parameter generator configuration. The system deploys $N = 3$ agents sharing the same backbone instance: Agent A analyzes reasoning types and plans chain-of-thought structure, Agent B retrieves domain knowledge and implicit constraints, and Agent C solves the problem after the generated LoRA perturbations are injected. Specific system prompts are designed for each role. The parameter generator is a lightweight Transformer ($d_{\text{pg}} = 1024$, head dimension 128, $N_{\text{pg}} = 2$ blocks, token dimension 256, dimension-accumulation factor $c = 2$) that produces LoRA factors of rank $r = 4$ with scaling $\alpha = 8$ ($s = \alpha/r = 2$).

Training details. The training data consist of OpenThoughts [Guha et al., 2025], Sky-T1 [Team, 2025], and KodCode [Xu et al., 2025]. We train for one epoch on 32,000 samples using a single NVIDIA RTX PRO 6000, which takes approximately 8 hours.

4.2 Main Results

Table 1 reports accuracy, average total processed token usage, and wall-clock inference time for TFlow, the Single-Agent baseline, and TextMAS across all five benchmarks.

Table 1: **Main results across five benchmarks.** For each task, we report accuracy (**Acc.** ↑, %), average total processed tokens (**Token** ↓), and end-to-end wall-clock inference time (**Speed**, in seconds). Single, TextMAS, and TFLOW are shown side by side. The **Improvement** columns compare TFLOW with each baseline: ↑/↓ denote accuracy increases/decreases in absolute points, ↓ denotes relative token reduction, and × denotes the speed ratio. Green indicates favorable changes, while red indicates unfavorable changes.

Tasks	Metrics	*Baselines*		TFLOW	Improvement	
		Single	TextMAS		Δ Single	Δ TextMAS
General Task						
MMLU-Redux	Acc. ↑	58.99	71.50	66.97	↑ **7.98**	↓ 4.53
	Token ↓	1079	4825	998	↓ **7.51%**	↓ **79.32%**
	Speed (s)	8226	36450	9784	×0.84	×**3.73**
Math & STEM Task						
GSM8K	Acc. ↑	84.99	93.78	92.12	↑ **7.13**	↓ 1.66
	Token ↓	1337	5381	900	↓ **32.69%**	↓ **83.27%**
	Speed (s)	6230	27256	5953	×**1.05**	×**4.58**
MATH	Acc. ↑	16.18	26.47	23.16	↑ **7.98**	↓ 3.31
	Token ↓	2782	8188	2242	↓ **19.41%**	↓ **72.62%**
	Speed (s)	1984	5213	2258	×0.88	×**2.31**
Coding Task						
MBPP+	Acc. ↑	59.79	68.52	67.20	↑ **7.41**	↓ 1.32
	Token ↓	1533	5500	1301	↓ **15.13%**	↓ **76.35%**
	Speed (s)	1998	5796	2395	×0.83	×**2.42**
Humaneval+	Acc. ↑	56.71	75.00	65.24	↑ **8.53**	↓ 9.76
	Token ↓	1756	5879	1662	↓ **5.35%**	↓ **71.73%**
	Speed (s)	872	2512	1065	×0.82	×**2.36**

Accuracy improvement over Single-Agent. TFLOW outperforms the single-agent baseline on every benchmark by $+6.98$ to $+8.53$ points, and does so with *fewer* processed tokens throughout, indicating that weight-space perturbations steer the receiver toward more concise reasoning rather than verbose exploration. Latency is nonetheless higher on four of the five benchmarks, since instance-specific perturbations prevent efficient batched generation.

Efficiency advantage over TextMAS. Weight-space injection bypasses the prefilling and KV-cache overhead of text concatenation, cutting processed tokens by 71–83% and yielding 2.3–4.6× wall-clock speed-ups, while the accuracy gap stays within 1.3–4.5 points on four of the five tasks. The sole outlier is HumanEval+ ($\Delta = 9.76$), where TextMAS's concatenated context plays a dual role: besides transferring knowledge, it implicitly extends the generation budget, enabling the longer outputs that help structural elaboration and self-debugging. TFLOW transfers collaborative knowledge but does not change output length, a distinction that surfaces most clearly in code generation.

4.3 Instance-level TFlow Analysis

A core property of TFLOW is that the LoRA consumed by the receiver is generated *per input*, rather than learned once and reused. We verify this by examining two points of the pipeline: the conditioning vector $\mathbf{c}$ that drives the parameter generator (section 4.3.1), and the resulting LoRA tensors for the receiver (section 4.3.2).

4.3.1 Hidden-State Evidence

We extract per-layer hidden states for 20 instances per dataset and report mean pairwise cosine similarity in three regimes: within-GSM8K, within-MBPP+, and cross-task (GSM8K × MBPP+).

Learned layer weights. As shown in Figure 3, the learned weights $\{w_\ell\}$ do *not* simply track the lowest pairwise-similarity layers. More than 80% of the mass concentrates on a small set of layers (L05, L29, L30), each with a distinct similarity profile. This selection emerges purely from end-to-end training, suggesting that these layers are chosen for their conditioning utility rather than surface-level diversity.

Pairwise similarity across regimes. Figure 3 further shows that the aggregated conditioning vector preserves the expected ordering of semantic similarity: within-GSM8K pairs (0.94) are most similar, followed by within-MBPP+ pairs (0.83), with cross-task pairs (0.73) being the least similar. This confirms that the conditioning representation encodes instance-level variation in a structured manner.

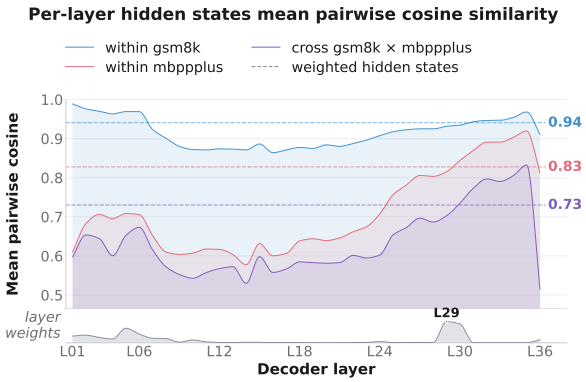


Figure 3: Per-layer mean **pairwise cosine similarity** of the question's last-token hidden state across decoder layers. Solid curves report within-task similarity and cross-task similarity, respectively. Dashed lines indicate the corresponding similarity computed on the aggregated conditioning vector $\mathbf{c}$ that the parameter generator actually consumes. The bottom panel shows the learned layer weights $\{w_\ell\}$, which peak sharply at layer 29.

4.3.2 LoRA Tensor Evidence

For each instance, we materialise the effective delta-weights $\Delta W_k = B_k A_k$ across all $K{=}252$ adapted projections (attention q, k, v, o and MLP $\mathrm{up}, \mathrm{gate}, \mathrm{down}$ on every decoder layer) and concatenate them into a single gauge-invariant fingerprint $\boldsymbol{\theta}$. We then compute the mean pairwise cosine similarity of $\boldsymbol{\theta}$ across $N{=}20$ instances on five benchmarks spanning three domains. Table 2 reports the resulting similarity matrix.

Distinct LoRAs per instance. The within-dataset similarity ranges from 0.264 to 0.933, reflecting the heterogeneity of the underlying instances: Python-coding tasks vary widely in specification, whereas MINERVA-MATH [Lewkowycz et al., 2022] problems share a highly templated structure. This spread indicates that TFLOW naturally allocates more adapter variation to datasets with diverse instances and less to near-templated ones. This is exactly the behaviour one would expect of an architecture that genuinely conditions on the input rather than collapsing to a fixed per-task adapter.

Domain-aware geometry. The off-diagonal entries exhibit clear block structure: within-domain pairs consistently show higher similarity than cross-domain pairs. The math block and the code block are the two highest cross-dataset entries, while the math $\times$ code crossings are the lowest. Notably, the within-dataset similarity of MBPP+ nearly coincides with its cross-similarity to HUMANEVAL+, which is expected given that both are Python function-level benchmarks with overlapping prompt formats. In all cases the diagonal dominates its row, confirming that $\boldsymbol{\theta}$ encodes task semantics faithfully.

4.4 Ablation Study: Instance-Specificity of Weight Perturbations

A key design choice in TFLOW is to generate *instance-specific* perturbations: the LoRA factors injected into the receiver are conditioned on both the sender's reasoning and the particular input. To validate that this instance-level conditioning is essential to the observed gains, we conduct the following ablations that progressively degrade specificity while holding all other variables constant.

Effect of the diversity regularizer. We also ablate the diversity regularizer $\mathcal{L}_{\mathrm{div}}$ (Section 3.5). Removing it lowers accuracy from 92.12 to 89.76 on GSM8K, from 67.20 to 62.17 on MBPP+, and from 65.24 to 59.76 on HumanEval+, a drop of 4.29 points on average (Table 3 in Appendix B.9). Without this objective, the parameter generator tends to produce overly similar perturbations for different inputs, which limits the benefit of sample-conditioned adaptation. Consistently, the gains are largest on the coding benchmarks, whose instances are the most heterogeneous (lowest within-dataset fingerprint similarity in Table 2).

Table 2: Mean pairwise cosine similarity of the gauge-invariant LoRA fingerprint $\boldsymbol{\theta}$ (N=20 instances per dataset). Diagonal: within-dataset; off-diagonal: cross-dataset.

	GSM8K	MINERVA-MATH	MMLU-REDUX	MBPP+	HUMANEVAL+
GSM8K	**0.519**	0.267	0.148	0.133	0.094
MINERVA-MATH	–	**0.933**	0.215	0.187	0.220
MMLU-REDUX	–	–	**0.424**	0.063	0.099
MBPP+	–	–	–	**0.264**	0.252
HUMANEVAL+	–	–	–	–	**0.315**

Static LoRA. To isolate instance-conditioned parameter generation from standard parameter-efficient adaptation, we replace the parameter generator of TFLOW with a conventional LoRA adapter shared across all inputs, while keeping the backbone, adapted modules, LoRA rank, and training data unchanged.

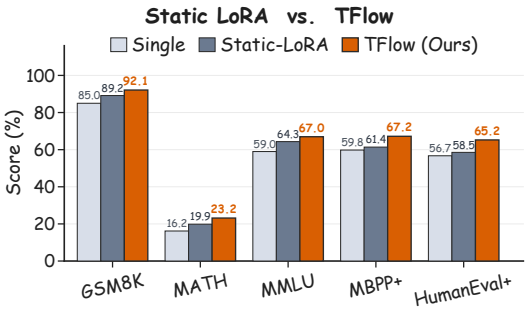


Figure 4: **Static LoRA vs. TFLOW** performance.

As shown in Figure 4, Static-LoRA consistently improves over the Single-Agent baseline on all benchmarks. However, TFLOW achieves substantially stronger performance, outperforming Static-LoRA by 4.29 points on average, with especially clear gains on more challenging reasoning and code-oriented benchmarks such as MBPP+ and HumanEval+. This shows that TFLOW's advantage goes beyond added trainable capacity and stems from input-dependent modulation of the receiver.

Mismatched Perturbation Injection. We further investigate whether TFLOW perturbations encode instance-level information by fixing the receiver input and replacing its matched perturbation with LoRA factors from random sources, other tasks, same-task samples, or the matched sample.

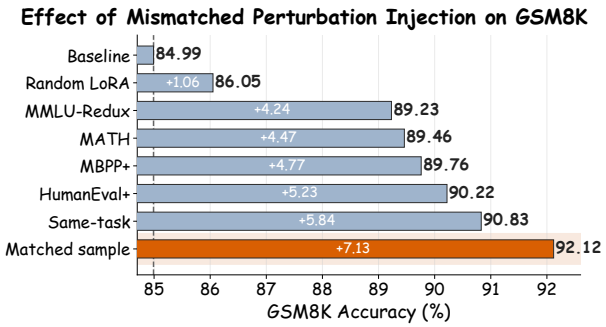


Figure 5: **Mismatched perturbation injection** on GSM8K.

Figure 5 shows that random LoRA perturbations bring only marginal gains, while cross-task perturbations still improve over the baseline, indicating that TFLOW captures transferable collaborative signals. However, same-task perturbations perform better, and the matched-sample perturbation achieves the highest accuracy, suggesting that TFLOW encodes not only task-level knowledge but also fine-grained instance-specific cues. This confirms that the generated perturbations are meaningfully tied to the target input, rather than serving as generic task-level or adapter-like updates.

5 CONCLUSION

We presented TFLOW, a weight-space collaboration paradigm for multi-agent LLM systems. Rather than routing inter-agent knowledge through natural language, TFLOW maps sender agents' hidden states into query-specific low-rank perturbations of a frozen receiver model. These perturbations are fused and transiently injected at inference time, enabling instance-level adaptation without increasing context length or requiring latent-space compatibility. Across five benchmarks, TFLOW consistently outperforms the single-agent baseline and achieves competitive accuracy with text-based multi-agent systems while using substantially fewer tokens and lower latency. Our findings indicate that model weights can serve as an effective communication medium among agents, suggesting a promising direction for efficient and scalable multi-agent LLM collaboration.

AI USE STATEMENT

In this work, we did not use generative AI tools for any task with required disclosure: they were not used to generate data, develop theoretical models or conceptual frameworks, formulate or prove mathematical claims, propose or refine hypotheses, design or provide feedback on the methodology or experiments, implement methods, clean or reformat data, support data analysis, or interpret results, and translation is not applicable to this work. Additionally, we used generative AI tools to edit the manuscript for readability. We have reviewed all AI-assisted work: all edits were read and approved by the authors, and every flagged citation was verified against the original paper before revision. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

All LLM backbones and benchmarks used in this work are publicly available, and the evaluation protocol, decoding settings, generator configuration, training data, and compute budget are described in Section 4.1. The architecture of the parameter generator, the sender fusion and transient injection procedure, the training objective, and the complete training and inference algorithms are specified in Appendix B. An anonymized code repository, including configurations, role prompts, and scripts to reproduce the main results, is provided as supplementary material; trained generator checkpoints will be released upon publication.

REFERENCES

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.

Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. Doc-to-lora: Learning to instantly internalize contexts. *arXiv preprint arXiv:2602.15902*, 2026.

Chih Yao Chen, Swarnadeep Saha, and Mohit Bansal. Reconcile: Round-table conference improves reasoning via consensus among diverse LLMs, 2024. URL https://openreview.net/forum?id=Yol6nUVIJD.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F Karlsson, Jie Fu, and Yemin Shi. Autoagents: A framework for automatic agent generation. *arXiv preprint arXiv:2309.17288*, 2023.

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.

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.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *ICML*, 2024.

Zhuoyun Du, Runze Wang, Huiyu Bai, Zouying Cao, Xiaoyong Zhu, Yu Cheng, Bo Zheng, Wei Chen, and Haochao Ying. Enabling agents to communicate entirely in latent space. *arXiv preprint arXiv:2511.09149*, 2025.

Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, et al. Are we done with MMLU? In *NAACL-HLT*, 2025.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, et al. Openthoughts: Data recipes for reasoning models. *arXiv preprint arXiv:2506.04178*, 2025.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

Xiaolong Han, Ferrante Neri, Zijian Jiang, Fang Wu, Yanfang Ye, Lu Yin, and Zehong Wang. W2t: Lora weights already know what they can do. *arXiv preprint arXiv:2603.15990*, 2026.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

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. *arXiv preprint arXiv:2103.03874*, 2021.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *ICLR*, 2023.

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 *ICML*, 2019.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *ICLR*, 2022.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. LoraHub: Efficient cross-task generalization via dynamic LoRA composition. In *COLM*, 2024.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*, 2022.

Haibo Jin, Kuang Peng, Ye Yu, Xiaopeng Yuan, and Haohan Wang. Agent primitives: Reusable latent building blocks for multi-agent systems. *arXiv preprint arXiv:2602.03695*, 2026.

Xiaolong Jin, Kai Wang, Dongwen Tang, Wangbo Zhao, Yukun Zhou, Junshu Tang, and Yang You. Conditional LoRA parameter generation. *arXiv preprint arXiv:2408.01415*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *SOSP*, 2023.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. In *NeurIPS*, 2023a.

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. More agents is all you need. *TMLR*, 2024.

Weishi Li, Yong Peng, Miao Zhang, Liang Ding, Han Hu, and Li Shen. Deep model fusion: A survey. *arXiv preprint arXiv:2309.15698*, 2023b.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent debate. In *EMNLP*, 2024.

Zhiyuan Liang, Dongwen Tang, Yuhao Zhou, Xuanlei Zhao, Mingjia Shi, Wangbo Zhao, Zekai Li, Peihao Wang, Konstantin Schürholt, Damian Borth, Michael M. Bronstein, Yang You, Zhangyang Wang, and Kai Wang. Drag-and-drop LLMs: Zero-shot prompt-to-weights. In *NeurIPS*, 2025.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *NeurIPS*, 2023.

Yewei Liu, Xiyuan Wang, Yansheng Mao, Yoav Gelbery, Haggai Maron, and Muhan Zhang. Shine: A scalable in-context hypernetwork for mapping context to lora in a single pass. *arXiv preprint arXiv:2602.06358*, 2026.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *ACL-IJCNLP*, 2021.

Michael S. Matena and Colin A. Raffel. Merging models with fisher-weighted averaging. In *NeurIPS*, 2022.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *UIST*, 2023.

Chau Pham, Boyi Liu, Yingxiang Yang, Zhengyu Chen, Tianyi Liu, Jianbo Yuan, Bryan A. Plummer, Zhaoran Wang, and Hongxia Yang. Let models speak ciphers: Multiagent debate through embeddings. In *ICLR*, 2024.

Jason Phang, Yi Mao, Pengcheng He, and Weizhu Chen. Hypertuning: Toward adapting large language models without back-propagation. In *ICML*, 2023.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In *MLSys*, 2023.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Xin Tang, Zhiyuan Li, Xuan Liu, Yuheng Yang, Yu Liang, Zilin Wang, Jinyang Yu, Pengyuan Zhou, Xu Xie, Wenhao Li, Yuhan Chen, and Cheng Li. ChatDev: Communicative agents for software development. In *ACL*, 2024.

Andries Petrus Smit, Nathan Grinsztajn, Paul Duckworth, Thomas D. Barrett, and Arnu Pretorius. Should we be going MAD? a look at multi-agent debate strategies for LLMs. In *ICML*, 2024.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Yichen Tang, Weihang Su, Yujia Zhou, Yiqun Liu, Min Zhang, Shaoping Ma, and Qingyao Ai. Augmenting multi-agent communication with state delta trajectory. In *EMNLP*, 2025.

NovaSky Team. Sky-t1: Train your own o1 preview model within $450. https://novasky-ai.github.io/posts/sky-t1, 2025. Accessed: 2025-01-09.

Tencent HY Team et al. Hy-wu (part i): An extensible functional neural memory framework and an instantiation in text-guided image editing. *arXiv preprint arXiv:2603.07236*, 2026.

Hanqing Wang, Bowen Ping, Shuo Wang, Xu Han, Yun Chen, Zhiyuan Liu, and Maosong Sun. LoRA-Flow: Dynamic LoRA fusion for large language models in generative tasks. In *ACL*, 2024a.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. In *ICLR*, 2025a.

Kai Wang, Dongwen Tang, Boya Zeng, Yida Yin, Zhaopan Xu, Yukun Zhou, Zelin Zang, Trevor Darrell, Zhuang Liu, and Yang You. Neural network diffusion. *arXiv preprint arXiv:2402.13144*, 2024b.

Kai Wang, Dongwen Tang, Wangbo Zhao, Konstantin Schürholt, Zhangyang Wang, and Yang You. Scaling up parameter generation: A recurrent diffusion approach. In *NeurIPS*, 2025b.

Zhiqi Wang, Shizhu He, Kang Liu, and Jun Zhao. Instance-level dynamic LoRAs composition for cross-task generalization. In *Findings of EMNLP*, 2024c.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *ICML*, 2022.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *COLM*, 2024.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101, 2025.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. KodCode: A diverse, challenging, and verifiable synthetic dataset for coding. In *Findings of ACL*, 2025.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. *NeurIPS*, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Enneng Yang, Li Shen, Guibing Guo, Xingwei Wang, Xiaochun Cao, Jie Zhang, and Dacheng Tao. Model merging in LLMs, MLLMs, and beyond: Methods, theories, applications, and opportunities. *ACM Computing Surveys*, 58(8):1–41, 2026.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *ICML*, 2024.

Yujia Zheng, Zhuokai Zhao, Zijian Li, Yaqi Xie, Mingze Gao, Lizhu Zhang, and Kun Zhang. Thought communication in multiagent collaboration. In *NeurIPS*, 2025.

Jiaru Zou, Xiyuan Yang, Ruizhong Qiu, Gaotang Li, Katherine Tieu, Pan Lu, Ke Shen, Hanghang Tong, Yejin Choi, Jingrui He, et al. Latent collaboration in multi-agent systems. *arXiv preprint arXiv:2511.20639*, 2025.

A LIMITATIONS

While TFLOW offers an efficient alternative to text-based multi-agent communication, it also has several limitations.

First, the communication channel is less interpretable than natural language. In text-based MAS, intermediate messages can be inspected by humans to understand what each agent contributed. In contrast, TFLOW transmits information through low-rank weight perturbations, whose semantic content is difficult to directly interpret. This may complicate debugging, attribution, and safety auditing, especially when the generated perturbations lead to unexpected receiver behavior.

Second, the performance gap on HumanEval+ indicates that TFLOW cannot fully recover the benefits that text-based MAS may obtain from increased generation length. Text-based collaboration not only transmits inter-agent information but also expands the overall reasoning and refinement budget through explicit intermediate rationales, plans, and deliberation traces. These additional tokens can directly benefit tasks such as code synthesis, where longer solutions, iterative correction, and detailed implementation planning are often important. TFLOW avoids passing sender-written messages to the receiver and therefore removes much of this extra generation overhead. This improves efficiency, but it also means that TFLOW may not capture all gains arising from the longer generation process used by TextMAS. Future work may explore hybrid systems that combine concise textual messages with transient parameter perturbations to balance interpretability, generation budget, and efficiency.

B Additional Method Details

This appendix provides additional details for TFlow that are omitted from the main text for brevity. We describe sender conditioning and layer aggregation, the parameter-generator architecture, sender fusion and transient receiver injection, the training objective, optimization algorithms, and computation cost.

B.1 Notation

We use N to denote the number of agents. Agents $A_1, \ldots, A_{N-1}$ are senders and A_N is the receiver. All agents share the same frozen backbone parameters $\boldsymbol{\theta}$ and are distinguished by role-specific prompts p_i. For a query q, sender A_i processes the prompted input (p_i, q) and produces hidden states $\{\mathbf{H}_i^{(l)}\}_{l=0}^{L}$, where

$$\mathbf{H}_i^{(l)} \in \mathbb{R}^{T_i \times d}.$$

Here T_i is the sender sequence length and d is the hidden dimension of the frozen backbone.

We use L for the number of targeted receiver layers and M for the number of targeted linear modules per targeted layer. The frozen weight of module m in layer l is denoted by

$$\mathbf{W}_m^{(l)} \in \mathbb{R}^{d_{\text{out}}^{(l,m)} \times d_{\text{in}}^{(l,m)}}.$$

The LoRA rank is r, and α is the LoRA scaling coefficient. The parameter generator has hidden dimension d_{pg} and N_{pg} transformer blocks.

B.2 Sender Conditioning and Layer Aggregation

For each query, every sender runs one frozen forward pass on its prompted input. We retain hidden states from all backbone layers:

$$\{\mathbf{H}_i^{(l)}\}_{l=0}^{L}, \qquad \mathbf{H}_i^{(l)} \in \mathbb{R}^{T_i \times d}.$$

Since different layers encode complementary levels of abstraction, we aggregate them using learnable layer weights. Let λ_l be a scalar assigned to layer l, and let $\tau > 0$ be a temperature. The normalized layer weight is

$$\rho_l = \frac{\exp(\lambda_l/\tau)}{\sum_{l'=0}^{L} \exp(\lambda_{l'}/\tau)}. \tag{4}$$

The sender conditioning signal is then

$$\mathbf{C}_i = \sum_{l=0}^{L} \rho_l \mathbf{H}_i^{(l)} \in \mathbb{R}^{T_i \times d}. \tag{5}$$

This construction lets the generator access information from both shallow and deep sender representations while preserving a fixed conditioning dimension.

Before entering the generator, the conditioning signal is projected to the generator hidden dimension:

$$\widetilde{\mathbf{C}}_i = \text{Proj}_{\text{in}}(\mathbf{C}_i) \in \mathbb{R}^{T_i \times d_{\text{pg}}}. \tag{6}$$

The sender backbone remains frozen, and gradients do not update $\boldsymbol{\theta}$.

B.3 Parameter Generator

The parameter generator $\mathcal{H}_\psi$ maps each sender conditioning signal $\mathbf{C}_i$ to LoRA factors for every targeted receiver module. For each layer $l \in \{1, \ldots, L\}$ and module $m \in \{1, \ldots, M\}$, it outputs

$$\mathbf{A}_{i,m}^{(l)} \in \mathbb{R}^{r \times d_{\text{in}}^{(l,m)}}, \qquad \mathbf{B}_{i,m}^{(l)} \in \mathbb{R}^{d_{\text{out}}^{(l,m)} \times r}. \tag{7}$$

Thus, the corresponding low-rank update has shape

$$\mathbf{B}_{i,m}^{(l)} \mathbf{A}_{i,m}^{(l)} \in \mathbb{R}^{d_{\text{out}}^{(l,m)} \times d_{\text{in}}^{(l,m)}}.$$

Token-grid initialization. The generator starts from a learnable token grid

$$\mathbf{Q}_{\text{grid}} \in \mathbb{R}^{L \times HW \times d_{\text{pg}}},$$

where $H = H_A + H_B$ separates tokens used for generating $\mathbf{A}$- and $\mathbf{B}$-factors, and W indexes module–rank slots. The projected sender condition $\widetilde{\mathbf{C}}_i$ initializes this grid through cross-attention:

$$\mathbf{Z}_i^{(0)} = \mathrm{CA}\big(\mathbf{Q}_{\text{grid}}, \widetilde{\mathbf{C}}_i, \widetilde{\mathbf{C}}_i\big) + \mathbf{E}_{\text{layer}} + \mathbf{E}_{\text{slot}}, \tag{8}$$

where $\mathbf{E}_{\text{layer}}$ encodes the target receiver-layer index and $\mathbf{E}_{\text{slot}}$ encodes module and rank-slot positions.

Multi-axis transformer trunk. The initialized grid is refined by N_{pg} transformer blocks. Each block contains cross-layer self-attention, intra-layer self-attention, conditioning cross-attention, and a feed-forward network. For block t, we write

$$\mathbf{U}_i^{(t)} = \mathbf{Z}_i^{(t)} + \mathrm{SA}_L\big(\mathrm{LN}(\mathbf{Z}_i^{(t)})\big), \tag{9}$$

$$\mathbf{V}_i^{(t)} = \mathbf{U}_i^{(t)} + \mathrm{SA}_{HW}\big(\mathrm{LN}(\mathbf{U}_i^{(t)})\big), \tag{10}$$

$$\mathbf{Y}_i^{(t)} = \mathbf{V}_i^{(t)} + \mathrm{CA}\big(\mathrm{LN}(\mathbf{V}_i^{(t)}), \widetilde{\mathbf{C}}_i, \widetilde{\mathbf{C}}_i\big), \tag{11}$$

$$\mathbf{Z}_i^{(t+1)} = \mathbf{Y}_i^{(t)} + \mathrm{FFN}\big(\mathrm{LN}(\mathbf{Y}_i^{(t)})\big). \tag{12}$$

Here, SA_L attends across targeted receiver layers while batching over the HW axis, and SA_{HW} attends within each layer across module–rank slots. The conditioning cross-attention re-grounds the parameter tokens in the sender representations at every block. RoPE is applied along the active attention axis.

Detokenization into LoRA factors. After the final generator block, the grid is split along the H axis into $\mathbf{A}$-tokens and $\mathbf{B}$-tokens:

$$\mathbf{Z}_i^{(N_{\text{pg}})} = [\mathbf{Z}_{i,A}, \mathbf{Z}_{i,B}].$$

Two linear heads project these tokens into parameter slices:

$$\mathbf{S}_{i,A} = h_A(\mathbf{Z}_{i,A}), \qquad \mathbf{S}_{i,B} = h_B(\mathbf{Z}_{i,B}). \tag{13}$$

The slices are rearranged according to layer, module, and rank-slot indices to produce

$$\left\{\mathbf{A}_{i,m}^{(l)}, \mathbf{B}_{i,m}^{(l)}\right\}_{l=1,m=1}^{L,M}.$$

All generator parameters are shared across senders. Sender-specific LoRA factors arise solely from sender-specific conditioning signals $\mathbf{C}_i$.

B.4 Sender Fusion and Transient Injection

When multiple senders are active, TFLOW fuses their generated updates at the perturbation level. This is important because fusing $\mathbf{A}$- and $\mathbf{B}$-factors separately would create unintended cross-sender products. Specifically,

$$\left(\sum_i \gamma_i \mathbf{B}_i\right)\left(\sum_j \gamma_j \mathbf{A}_j\right) = \sum_{i,j} \gamma_i \gamma_j \mathbf{B}_i \mathbf{A}_j,$$

which includes terms $\mathbf{B}_i\mathbf{A}_j$ for $i \neq j$. We therefore fuse complete low-rank perturbations $\mathbf{B}_i\mathbf{A}_i$.

The sender-fusion gate predicts one scalar score for each sender:

$$s_i = g_{\mathbf{w}}(\mathbf{C}_i). \tag{14}$$

Scores are normalized across active senders:

$$\gamma_i = \frac{\exp(s_i)}{\sum_{j=1}^{N-1} \exp(s_j)}. \tag{15}$$

The fused transient update for module m in layer l is

$$\Delta \mathbf{W}_m^{(l)} = \frac{\alpha}{r} \sum_{i=1}^{N-1} \gamma_i \mathbf{B}_{i,m}^{(l)} \mathbf{A}_{i,m}^{(l)}. \tag{16}$$

For a receiver activation matrix $\mathbf{X} \in \mathbb{R}^{T \times d_{\text{in}}^{(l,m)}}$, the patched linear map is

$$\mathbf{X} \mapsto \mathbf{X}(\mathbf{W}_m^{(l)})^\top + \mathbf{X}(\Delta \mathbf{W}_m^{(l)})^\top. \tag{17}$$

The dense matrix $\Delta \mathbf{W}_m^{(l)}$ need not be explicitly materialized. The additive branch can be computed in low-rank form:

$$\mathbf{X}(\Delta \mathbf{W}_m^{(l)})^\top = \frac{\alpha}{r} \sum_{i=1}^{N-1} \gamma_i \big(\mathbf{X}(\mathbf{A}_{i,m}^{(l)})^\top\big)(\mathbf{B}_{i,m}^{(l)})^\top. \tag{18}$$

The frozen weight tensor $\mathbf{W}_m^{(l)}$ is never overwritten. The update exists only during the current receiver forward pass and is removed immediately afterward.

B.5 Training Objective

The trainable parameter set is

$$\Theta = \{\boldsymbol{\psi}, \text{Proj}_{\text{in}}, \{\lambda_l\}_{l=0}^{L_{\text{total}}}, \mathbf{w}\},$$

where $\boldsymbol{\psi}$ denotes the parameter-generator weights, Proj_{in} denotes the conditioning projection, $\{\lambda_l\}$ are the layer-aggregation scalars, and $\mathbf{w}$ denotes the fusion-gate parameters. The frozen backbone parameters $\boldsymbol{\theta}$ are excluded from Θ.

Task loss. For a training tuple (x_q, t, σ), where x_q is the query-side input, t is the target completion, and σ is the data-source identifier, the task loss is the masked next-token cross-entropy over completion tokens:

$$\mathcal{L}_{\text{task}}(\Theta; x_q, t) = -\sum_{u \in \mathcal{I}_{\text{ans}}} \log p_{\boldsymbol{\theta},\Theta}\big(t_u \mid t_{<u}, x_q\big), \tag{19}$$

where $\mathcal{I}_{\text{ans}}$ denotes answer-token positions. The distribution $p_{\boldsymbol{\theta},\Theta}$ is computed by the frozen receiver equipped with the transient updates produced by Θ.

Flattened generated update. To define the diversity regularizer, we flatten all fused updates for the current input:

$$v_\Theta(x_q) = \text{vec}\left(\left\{\Delta \mathbf{W}_m^{(l)}(x_q)\right\}_{l=1,m=1}^{L,M}\right). \tag{20}$$

In implementation, this vector can be represented either by explicitly flattened dense offsets or by an equivalent flattened low-rank representation, as long as the same representation is used consistently for the current vector and the cache.

Cache-based diversity regularizer. For each data source σ, we maintain a stop-gradient cache vector $v_{\text{prev}}^{(\sigma)}$, corresponding to the generated update from the previous optimization step involving that source. The diversity regularizer is

$$\mathcal{L}_{\text{div}}(\Theta; x_q) = \cos^2\big(v_\Theta(x_q), v_{\text{prev}}^{(\sigma(x_q))}\big). \tag{21}$$

This objective penalizes both parallel and anti-parallel collapse and is minimized when the current update is orthogonal to the cached update. Since $v_{\text{prev}}^{(\sigma)}$ is detached, the regularizer does not introduce an additional forward pass.

The total objective is

$$\mathcal{L}(\Theta) = \mathbb{E}_{(x_q,t,\sigma)\sim\mathcal{D}}\left[\mathcal{L}_{\text{task}}(\Theta; x_q, t) + \lambda_{\text{div}} \mathcal{L}_{\text{div}}(\Theta; x_q)\right]. \tag{22}$$

Algorithm 1 Training TFLOW

Require: Frozen backbone $f_{\boldsymbol{\theta}}$; trainable parameters Θ; sender prompts $\{p_i\}_{i=1}^{N-1}$; receiver prompt p_N; dataset $\mathcal{D}$; diversity weight λ_{div}.

Initialize diversity cache $\{v_{\text{prev}}^{(\sigma)}\}$ for each data source σ.
for each minibatch $\mathcal{B} \subset \mathcal{D}$ **do**
  Initialize minibatch loss $\mathcal{L}_{\mathcal{B}} \leftarrow 0$.
  for each example $(x_q, t, \sigma) \in \mathcal{B}$ **do**
    for each sender $i = 1, \ldots, N-1$ **do**
      Run frozen sender forward on (p_i, x_q) and collect $\{\mathbf{H}_i^{(l)}\}_{l=0}^{L_{\text{total}}}$.
      Aggregate hidden states into $\mathbf{C}_i$ using Equation (5).
      Generate LoRA factors $\{\mathbf{A}_{i,m}^{(l)}, \mathbf{B}_{i,m}^{(l)}\}_{l,m} \leftarrow \mathcal{H}_{\boldsymbol{\psi}}(\mathbf{C}_i)$.
    end for
    Compute gate weights $\gamma_i = \text{softmax}_i(g_{\mathbf{w}}(\mathbf{C}_i))$.
    Fuse sender updates into $\Delta\mathbf{W}_m^{(l)}$ for all targeted modules using Equation (16).
    Temporarily patch the targeted receiver modules.
    Run the frozen receiver on (p_N, x_q, t) and compute $\mathcal{L}_{\text{task}}$ over completion tokens.
    Form $v_\Theta(x_q)$ and compute $\mathcal{L}_{\text{div}}$ using the stop-gradient cache $v_{\text{prev}}^{(\sigma)}$.
    Accumulate $\mathcal{L}_{\mathcal{B}} \leftarrow \mathcal{L}_{\mathcal{B}} + \mathcal{L}_{\text{task}} + \lambda_{\text{div}}\mathcal{L}_{\text{div}}$.
    Remove the temporary receiver patches.
  end for
  Update Θ using gradients of $\mathcal{L}_{\mathcal{B}}$.
  Update each used cache entry $v_{\text{prev}}^{(\sigma)} \leftarrow \text{stopgrad}(v_\Theta(x_q))$.
end for

B.6 Training Algorithm

B.7 Inference Algorithm

At inference time, TFLOW follows the same sender-conditioning and parameter-generation process, but omits the task loss, diversity regularizer, and cache update. The generated perturbation is used only during receiver decoding.

Algorithm 2 Inference with TFLOW

Require: Frozen backbone $f_{\boldsymbol{\theta}}$; trained parameters Θ; sender prompts $\{p_i\}_{i=1}^{N-1}$; receiver prompt p_N; query q.

for each sender $i = 1, \ldots, N-1$ **do**
  Run frozen sender forward on (p_i, q) and collect $\{\mathbf{H}_i^{(l)}\}_{l=0}^{L_{\text{total}}}$.
  Aggregate hidden states into $\mathbf{C}_i$.
  Generate LoRA factors $\{\mathbf{A}_{i,m}^{(l)}, \mathbf{B}_{i,m}^{(l)}\}_{l,m} \leftarrow \mathcal{H}_{\boldsymbol{\psi}}(\mathbf{C}_i)$.
end for
Compute $\gamma_i = \text{softmax}_i(g_{\mathbf{w}}(\mathbf{C}_i))$.
Fuse updates into $\Delta\mathbf{W}_m^{(l)} = \frac{\alpha}{r}\sum_{i=1}^{N-1}\gamma_i\mathbf{B}_{i,m}^{(l)}\mathbf{A}_{i,m}^{(l)}$.
Temporarily patch the targeted receiver modules.
Decode the final answer using receiver A_N conditioned on (p_N, q).
Remove all temporary patches.

B.8 Computation Cost

The additional cost of TFLOW beyond a standard receiver forward pass comes from sender forwarding, parameter generation, and transient LoRA injection.

Sender forward passes. Let $C_{\mathrm{bb}}(T)$ denote the cost of one frozen backbone forward pass with sequence length T. The $N-1$ senders incur

$$C_{\mathrm{sender}} = \sum_{i=1}^{N-1} C_{\mathrm{bb}}(T_i). \tag{23}$$

These sender passes are independent and can be parallelized. They also do not require autoregressive decoding; each sender only produces hidden states for conditioning.

Parameter generation. Let $S = HW$ be the number of parameter tokens per targeted layer. Each generator block applies cross-layer attention, intra-layer attention, conditioning cross-attention, and an FFN. For sender i, the cross-layer attention costs $O(SL^2 d_{\mathrm{pg}})$, the intra-layer attention costs $O(LS^2 d_{\mathrm{pg}})$, the conditioning cross-attention costs $O(LST_i d_{\mathrm{pg}})$, and the FFN costs $O(LSd_{\mathrm{pg}}^2)$. Therefore,

$$C_{\mathrm{gen}}^{(i)} = O\big(N_{\mathrm{pg}}\left[SL^2 d_{\mathrm{pg}} + LS^2 d_{\mathrm{pg}} + LST_i d_{\mathrm{pg}} + LSd_{\mathrm{pg}}^2\right]\big), \tag{24}$$

and across all senders,

$$C_{\mathrm{gen}} = \sum_{i=1}^{N-1} C_{\mathrm{gen}}^{(i)}. \tag{25}$$

Transient LoRA injection. For a targeted linear module with receiver sequence length T, input dimension $d_{\mathrm{in}}^{(l,m)}$, output dimension $d_{\mathrm{out}}^{(l,m)}$, and LoRA rank r, the low-rank branch in Equation (18) costs

$$O\Big((N-1)Tr\left(d_{\mathrm{in}}^{(l,m)} + d_{\mathrm{out}}^{(l,m)}\right)\Big). \tag{26}$$

This is small compared with the dense projection cost

$$O\Big(Td_{\mathrm{in}}^{(l,m)} d_{\mathrm{out}}^{(l,m)}\Big)$$

when $r \ll \min(d_{\mathrm{in}}^{(l,m)}, d_{\mathrm{out}}^{(l,m)})$. Summing over all targeted modules gives

$$C_{\mathrm{inj}} = O\left((N-1)Tr\sum_{l=1}^{L}\sum_{m=1}^{M}\left(d_{\mathrm{in}}^{(l,m)} + d_{\mathrm{out}}^{(l,m)}\right)\right). \tag{27}$$

Overall overhead. The total additional cost is

$$C_{\mathrm{extra}} = C_{\mathrm{sender}} + C_{\mathrm{gen}} + C_{\mathrm{inj}}. \tag{28}$$

The sender and generator costs scale linearly with the number of active senders $N-1$, and the injection cost scales linearly with both $N-1$ and the LoRA rank r. In practice, sender forward passes and per-sender generation are independent and can be batched or parallelized, reducing wall-clock overhead.

B.9 Ablation on the Diversity Regularizer

Table 3 reports the accuracy of TFlow trained with and without the diversity regularizer $\mathcal{L}_{\mathrm{div}}$ of Section 3.5, with all other settings unchanged. Introducing $\mathcal{L}_{\mathrm{div}}$ improves accuracy by 2.36 points on GSM8K, 5.03 points on MBPP+, and 5.48 points on HumanEval+, an average improvement of 4.29 points. Without this objective, the parameter generator may produce overly similar perturbations for different inputs, which limits the benefit of sample-conditioned adaptation.

Table 3: **Ablation on the diversity regularizer $\mathcal{L}_{\mathrm{div}}$.** Accuracy (%) with a Qwen3-4B backbone.

Configuration	GSM8K	MBPP+	HumanEval+	Avg.
TFlow without $\mathcal{L}_{\mathrm{div}}$	89.76	62.17	59.76	70.56
TFlow with $\mathcal{L}_{\mathrm{div}}$ (default)	**92.12**	**67.20**	**65.24**	**74.85**