Self-Awareness First

Riccardo Alberghi[1] Alessandro Bellini[1] Flavio Caroli[1] Luca Colaci[1] Matilde Dolfato[1]

[1]Bocconi Students for Machine Learning, Bocconi University, Milan, Italy
{riccardo.alberghi alessandro.bellini flavio.caroli luca.colaci
matilde.dolfato}@studbocconi.it

October 14, 2025

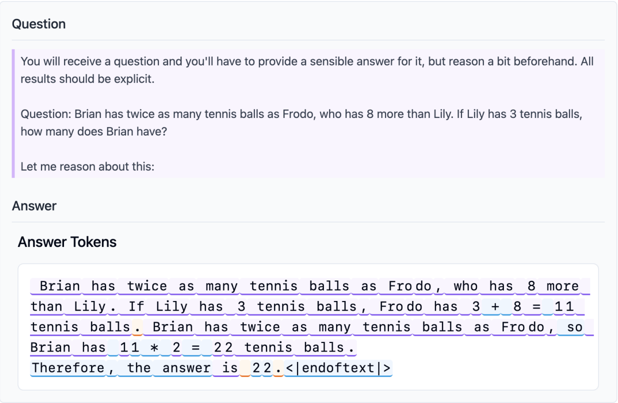
Abstract

We introduce a new approach to Large Language Model (LLM) ensembling through dynamic routing of token generation. Instead of conventional ensemble methods that blend model outputs, we propose a token-by-token routing mechanism where each token in the response is generated by the most suitable model within the ensemble. We augment each base model with a *self-awareness* scoring head that predicts the model's suitability for generating the next token in the sequence. This enables dynamic expert selection during generation. Our experiments compare two model families— Llama-3.2 (1B parameters) and Qwen 2.5 (1.5B parameters) —including general-purpose, mathematics-specific, and medical-specific fine-tuning, across GSM8K and MedQA benchmarks. Results show that our ensemble method **saLM-1** effectively leverages complementary strengths among specialized models, particularly with the Qwen architecture. Here, the ensemble achieves 20% accuracy on MedQA, while matching the 80% performance of the mathematics specialist on GSM8K. We examine both linear and non-linear routing heads and discuss the challenges in balancing expert utilization. This work contributes to the emerging field of LLM ensembling by providing a framework for context-aware model selection at the token level, potentially offering more precise and domain-appropriate responses than specialized models alone.

1 Introduction

The Large Language Model (LLM) industry has recently been following two parallel paths for augmenting models' capabilities: improving the architecture and scaling capacity [9]. While the former approach requires novelty, the latter needs ever-increasing computational resources.

In the realm of industrial applications, LLMs are usually employed for vertical tasks requiring advanced specific knowledge and strict adherence to company guidelines. To enforce these rules in a systematic manner, model fine-tuning is one of the best practices, thanks to the availability of additional related data to feed the model [12].



Question

You will receive a question and you'll have to provide a sensible answer for it, but reason a bit beforehand. All results should be explicit.

Question: Brian has twice as many tennis balls as Frodo, who has 8 more than Lily. If Lily has 3 tennis balls, how many does Brian have?

Let me reason about this:

Answer

Answer Tokens

```
 Brian has twice as many tennis balls as Fro do , who has 8 more
than Lily . If Lily has 3 tennis balls , Fro do has 3 + 8 = 11
tennis balls . Brian has twice as many tennis balls as Fro do , so
Brian has 11 * 2 = 22 tennis balls .
Therefore , the answer is 22 .<|endoftext|>
```

Figure 1: A generation example of the saLM-1 model on a GSM8K problem. Purple tokens: general model; orange: Med; blue: Math.

Alongside this, the industry is exploring the direction of LLM ensembles [1], involving the comprehensive use of multiple LLMs. This is a widely used technique in classical machine learning that is the state of the art in Natural Language Processing (NLP) applications [5], [1] .

In this work, we propose saLM-1, a novel LLM-

ensembling method that actively routes each output token generation to the model whose *self-awareness score* is highest. Our hypothesis is that through this technique we can make each model generate the most appropriate tokens according to the specific tasks it was trained on, hence producing better quality output.
To this end, we first augment base models' architecture with a simple linear projection layer, and then with a multi-layer perceptron.
The idea is that with a projection or non-linear transformation of the enriched representation of the last token in the sequence, each model will learn if it is *in distribution* with respect to the context, and able to correctly generate the next token, or if it is *out of distribution* and generation should be left to other models in the ensemble. All the code pertinent to this project can be found on GitHub [1].

2 Related Work

Our work extends the collaborative-decoding paradigm by [10]. In Co-LLM, a large "manager" model treats the choice of the generator — between itself and several smaller, domain-specialized "assistant" LLMs — as a latent variable at every time-step. By maximizing the marginal likelihood of the training corpus, the manager learns without direct supervision when to emit a token on its own and when to delegate. The resulting system behaves like an oracle-guided ensemble: decisions are centrally coordinated, and there is built-in asymmetry because the manager is substantially larger than the assistants.
In contrast, we investigate peer-level collaboration. All participating LLMs have the same parameter count and identical Transformer architecture (though they differ in fine-tuned expertise). Without a privileged controller, the models must jointly discover cooperative policies rather than relying on top-down delegation. This shift from hierarchical to symmetric collaboration forces the ensemble to develop emergent negotiation strategies, offering a cleaner test-bed for studying truly collaborative decoding and opening the door to settings where no single model can act as an oracle.

Mixture of Experts, MoE The Switch Transfomer architecture proposed in [3] shares some similarities with our work, especially in the use of a routing mechanism. We deem it appropriate to clarify the distinction. Switch Transformers scale capacity by inserting a gated layer inside each Transformer block, that activates one of many feed-forward 'experts' per token. This produces a single sparse network that must be trained end-to-end, and whose experts are indistinguishable outside the gating context.
Instead, our setting keeps several fully realized LLMs of equal size, trained (or fine-tuned) independently, and combined only at decoding time. Collaboration therefore happens across separate models rather than between hidden sub-modules, and no global sparsity gate or joint training phase is required.

3 Methodology

3.1 Primitive Models

Our models are from two architectures: Llama-3.2 (1B parameters) by Meta [6] and Qwen-2.5 (1.5B) by Alibaba Cloud [11].
These are chosen as they balance generation quality and dimension. For each of the two, we select three models (all coming from the HuggingFace community): one is a general-purpose model, one is fine-tuned for mathematics applications and one for medical applications:

- General-purpose: bunnycore/Llama-3.2-1B-General-lora_model and Qwen/Qwen2.5-1.5B
- Math-specific: axel-datos/Llama-3.2-1B_gsm8k_full-finetuning and Qwen/Qwen2.5-Math-1.5B
- Med-specific: Johhny1201/llama3.2_1b_med_QA_2 and Arthur-77/QWEN2.5-1.5B-medical-finetuned

Our model choice is most importantly tied to the models' tokenizers. Our current architecture can only deal with minor changes between tokenizers, as we pick one to encode for all models (a deeper dive in the matter can be found in Section 4).
Several additional factors determine our choice of models, such as availability, nature and quality of the datasets used for fine-tuning and performance.
Also, the generation quality of the ensemble model heavily relies on the capabilities of the primitive models, but the computational resources at our disposal also constrain it.

3.2 Training Data

We leverage publicly available datasets on HuggingFace. First, we process the GSM8K dataset

[1] https://github.com/riccardoalberghi/saLM-1.git

[2], a large collection of high school-level math problems with extensive reasoning provided together with answers. The dataset was formatted in a 2-column manner, one providing the problem and the other providing the reasoning and the final answer.

We perform a similar preprocessing on MedQA [7], a large collection of multiple-choice questions taken from the medical board exams in English. Again, the dataset is formatted in a `question` column, containing the question and available options, and an `answer` column, containing the correct answer. Both datasets are publicly available on HuggingFace at cola13/medqa_formatted and cola13/gsm8k_formatted.

Notice that our format provides explicit introductions to the parts of the question/answer by the addition of "*Question: ...*", "*Options: ...*" and "*Reasoning: ...*" markers, as well as a more formal introduction to the correct answer through the formula "*The correct answer is: ...*". This is done to provide the models with more context.

4 saLM-1 : Self-Aware Language Models (Ensemble)

saLM-1 is an **ensemble of pre-trained language models** sharing the same architecture but differing in their fine-tuning. To maintain consistency in tokenization and label alignment across the ensemble, all submodels employ a single, shared tokenizer with a common vocabulary, as mentioned above. We only integrate models whose backbones and vocabularies coincide because a uniform token set is essential for computing our joint loss. As we will explain later, this loss relies on a weighted aggregation of each submodel's token probabilities, which would be ill-defined if their vocabularies diverged. Consult Appendix A to inspect the full tokenizer matching analysis.

Each model's architecture is augmented by attaching a head that learns the model's self-awareness of its suitability for predicting the next token. Ideally, at each decoding step, we dynamically select the submodel that is most likely to be *in distribution* for the current context.

4.1 Self-Awareness Score

Let $\mathcal{M} = \{\mathbf{M}_1, \ldots, \mathbf{M}_M\}$ denote our set of M submodels. For each submodel $\mathbf{M}_m$, we compute a *self-awareness score* $\alpha_m[b,t] \in \mathbb{R}$ at batch index b and token position t. This score estimates the probability that $\mathbf{M}_m$ is *in-distribution* and therefore reliable for next-token prediction. Crucially, this differs from taking the highest softmax probability within a model's own vocabulary distribution: a high softmax value may reflect overconfidence on out-of-distribution inputs, whereas α_m explicitly predicts whether the model "knows" the current input.

4.2 Head Architecture

A *ScalarProjectionHead* is attached to each model's last hidden layer (dimension H), producing the raw score $s_m[b,t]$. We train two types of heads, one non-linear and a linear one.

The linear head consists of just a layer $\mathbb{R}^H \to \mathbb{R}^1$, producing $s_m[b,t]$.

The non-linear head consists of:

1. **First projection:** a linear layer $\mathbb{R}^H \to \mathbb{R}^D$ (with $D = 256$ by default), then batch normalization, ReLU activation, and dropout.
2. **Second projection:** a linear layer $\mathbb{R}^D \to \mathbb{R}^{D/2}$ (default 128), then batch normalization, ReLU activation, and dropout.
3. **Final scalar output:** a linear layer $\mathbb{R}^{D/2} \to \mathbb{R}^1$, producing $s_m[b,t]$.

Each head is trained jointly with the routing network to predict per-token self-awareness, enabling dynamic expert selection during generation.

4.3 Routing (Switching) Method

At each time step (b,t), we compute raw self-awareness logits $\{s_m[b,t]\}_{m=1}^{M}$ from each submodel's projection head (see Section 4.2). We normalize these via

$$p_m[b,t] \;=\; \frac{\exp(s_m[b,t])}{\sum_{m'=1}^{M} \exp(s_{m'}[b,t])}\,,$$

yielding a categorical distribution over $\mathcal{M}$. The submodel with the highest $p_m[b,t]$ is selected to emit the next token.

4.4 Loss function

To train the head we use the binary cross-entropy loss between the true token distribution and a combined estimated distribution. This is the weighted average of the distributions estimated by each base model, with weights being pre-normalization self-awareness levels for each base model.

Let M be the number of submodels. For each input batch of size B and sequence length T, define:

- $\mathbf{p}_m \in \mathbb{R}^{B\times T}$: routing probability for submodel $\mathbf{M}_m$, with $\sum_{m=1}^{M} p_m[b,t] = 1$ for all b, t.
- $\mathbf{z}_m \in \mathbb{R}^{B\times T\times V}$: per-token softmax probabilities from submodel $\mathbf{M}_m$ over a vocabulary of size V.
- $y \in \{1,\dots,V\}^{B\times T}$: ground-truth token IDs, with ignore index γ marking padding or masked positions.

We form the *mixture distribution* over tokens:

$$\hat{P}(x_t = v \mid b,t) \;=\; \sum_{m=1}^{M} p_m[b,t]\; z_m[b,t,v].$$

The negative log-likelihood (NLL) over non-ignored positions yields:

$$\begin{aligned} L_{\text{NLL}} = &-\frac{1}{\sum_{b,t} \mathbb{1}[y_{b,t} \neq \gamma]} \\ &\times \sum_{b,t:y_{b,t}\neq\gamma} \log\big(\hat{P}(x_t = y_{b,t} \mid b,t)\big) \end{aligned}$$

To promote balanced expert usage, we add an entropy regularizer on the routing distributions:

$$\begin{aligned} H(\mathbf{p}) = &-\frac{1}{\sum_{b,t} \mathbb{1}[y_{b,t} \neq \gamma]} \\ &\times \sum_{b,t:y_{b,t}\neq\gamma} \sum_{m=1}^{M} p_m[b,t] \log p_m[b,t]. \end{aligned}$$

The combined loss is:

$$L = L_{\text{NLL}} \;-\; \lambda\, H(\mathbf{p}),$$

where λ controls the strength of entropy regularization. In practice, we add a small ε (e.g., 10^{-10}) inside logarithms for numerical stability. If a batch contains only ignored positions, we define $L = 0$ to preserve gradient flow without affecting the objective.

4.5 Training process

We train the mixture model end-to-end with gradient-based methods, optimizing the Perfect Alignment Loss. Each epoch proceeds as follows:

1. **Data Preparation:** we tokenize input phrases with a common tokenizer to produce input IDs $\mathbf{x} \in \mathbb{Z}^{B\times T}$. Then, we construct next-token labels by right-shifting: $\mathbf{y}_{b,t} = \mathbf{x}_{b,t+1}$ for $t < T$, and set $\mathbf{y}_{b,T} = \gamma$. Finally, we mask padding positions to γ.
2. **Forward Pass:** first, each submodel m computes logits $\ell_m \in \mathbb{R}^{B\times T\times V}$, then sets $\mathbf{z}_m = \text{softmax}(\ell_m)$. After, each routing head produces raw scores $s_m \in \mathbb{R}^{B\times T}$, setting $\mathbf{p}_m = \text{softmax}(\{s_1,\dots,s_M\})$ across m.
3. **Loss Computation:** we align sequence lengths and mask positions to ignore, then compute L as above.
4. **Backward Pass & Update:** we backpropagate through submodels and routing head, then update parameters via an AdamW optimizer [8].
5. **Logging & Validation:** every N steps, we log the training loss, perplexity, and routing statistics, more precisely:
   - *Activation counts*: frequency each submodel is most probable.
   - *Mean routing probability*: average p_m across tokens.

4.6 Evaluation methods

We test the ensemble model on a subset of the datasets. For each example, the ensemble model is provided with the `question` column of the dataset, and asked to reason about task and provide an answer starting with a specific sentence (e.g. "The answer is"). We perform prompt engineering with a trial-and-error approach (the final prompt is visible in the code). This is an attempt to force the prompt into generating a Chain of Thought (CoT), as it is shown that it allows the model to generate more confident and, in general, better answers [4].
More specifically, the questions in the GSM8K dataset require the model to understand the problem, break it into steps, and solve it, providing the final answer. We evaluate this task through the `math_verify` library, provided by HuggingFace to evaluate LLMs on math reasoning. By providing the library with the model-generated answer and the ground truth, we obtain the correctness of the model's output.
For the MedQA dataset, we prompt the models with questions and the list of possible answers, indexed by letters. Then, the models have to reason about the question and provide both the answer and the associated letter. We parse the output through a custom Regex, matching a variety of possible answer formats, including deviations from the required one. Finally, we confront the scraped answer with the ground truth to obtain the accuracy score of the model.

4.7 Differences between generation and training

Let us clarify a critical difference between the processes of training and generation. When training, we create a combined token probability distribution, due to constraints given by the form of our loss function. Then, the token to be generated is sampled from this combined probability. On the other hand, during generation, we switch between models by taking the `argmax` over self-awareness scores.

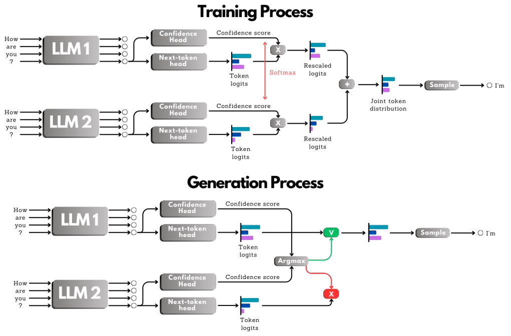


Figure 2: Differences between behavior during training vs generation.

5 Results and Discussion

We evaluate models on 50 questions from the training set and test each model separately.

5.1 Llama with non-linear heads

Table 1: Accuracy of Llama-3 Ensemble and Individual Models on GSM8K and MedQA Benchmarks

Model	MedQA	GSM8K
saLM-1	12.0%	4.0%
Llama-3 2-1B	**22.0%**	**28.0%**
Llama-3 2-1B Math	6.0%	2.0%
Llama-2-1B Med	6.0%	4.0%

Since we select the base model with the highest self-awareness score at each token — rather than blending their output distributions - the ensemble should route to the best expert. In practice, noisy or miscalibrated self-awareness estimates can still cause weaker specialists to be chosen, degrading overall performance. This observation underscores the need for sharper self-awareness calibration or task-specific routing priors to bias selection toward consistently high-performing models. Moreover, pruning the ensemble to include only models with more parameters and understanding may further improve results.

5.2 Qwen

Table 2: Accuracy of Qwen Ensemble and Individual Models on MedQA and GSM8K

Model	MedQA	GSM8K
saLM-1 (L)	14.0%	**80.0%**
saLM-1 (NL)	20.0%	78.0%
qwen-25-1.5b	18.0%	68.0%
qwen-25-1.5b-med	**20.0%**	72.0 %
qwen-25-1.5b-math	14.0%	**80.0%**

We evaluated the Qwen ensemble using both a linear and non-linear head (the differences are explained in Section 4.2). Both variants fall in the middle range of performance for the MedQA task, achieving performances as bad as the worst base model and being outclassed by the fine-tuned version. On the GSM8K task, they attain 78 and 80 %, equivalent to the best single model (the math-fine-tuned variant at 80 %) and surpassing the others. This provides evidence that our dynamic routing effectively leverages complementary strengths: the ensemble generalizes better on the task where no single model dominates, and still competes with the specialist on its home turf.

6 Conclusions

Our work has explored a new way of ensembling LLMs through routing token generation between base models, creating saLM-1. Overall, our findings indicate that it is possible to make models aware of their own generation capabilities in a collaborative environment. On the two tested tasks, saLM-1 proves capable of matching its fine-tuned components' performance, leveraging dynamic token-by-token routing based on self-awareness.

Our research is limited in that the computational power available to us allowed for no more than three models in the ensemble. Also, future research on the matter should investigate the possibility of sharing the KV cache for ensembles based on the same architecture and better integration of tokenizers with different vocabularies.

References

[1] Zhijun Chen, Jingzheng Li, Pengpeng Chen, Zhuoran Li, Kai Sun, Yuankai Luo, Qianren Mao, Dingqi Yang, Hailong Sun, and Philip S. Yu. Harnessing multiple large language models: A survey on llm ensemble, 2025.

[2] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, et al. Training verifiers to solve math word problems, 2021.

[3] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2021.

[4] Tairan Fu, Javier Conde, Gonzalo Martínez, María Grandury, and Pedro Reviriego. Multiple choice questions: Reasoning makes large language models (llms) more self-confident even when they are wrong, 2025.

[5] M. A. Ganaie, Minghui Hu, A. K. Malik, M. Tanveer, and P. N. Suganthan. Ensemble deep learning: A review. *Engineering Applications of Artificial Intelligence*, 115:105151, 2022.

[6] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, et al. The llama 3 herd of models, 2024.

[7] Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams, 2020.

[8] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

[9] Mohaimenul Azam Khan Raiaan, Md. Saddam Hossain Mukta, Kaniz Fatema, Nur Mohammad Fahad, Sadman Sakib, Most Marufatul Jannat Mim, Jubaer Ahmad, Mohammed Eunus Ali, and Sami Azam. A review on large language models: Architectures, applications, taxonomies, open issues and challenges. *IEEE Access*, 12:26839–26874, 2024.

[10] Shannon Zejiang Shen, Hunter Lang, Bailin Wang, Yoon Kim, and David Sontag. Learning to decode collaboratively with multiple language models, 2024.

[11] Qwen Team, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, et al. Qwen2.5 technical report, 2025.

[12] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners, 2022.

A Appendix A: Tokenizer and Vocabulary Statistics

A.1 Loaded Tokenizers

- `qwen-25-15b` (Qwen/Qwen2.5-1.5B), vocab size: 151 665
- `quen-25-15b-med` (Arthur-77/QWEN2.5-1.5B-medical-finetuned), vocab size: 151 665
- `qwen-25-15b-math` (Qwen/QWEN2.5-Math-1.5B), vocab size: 151 665

A.2 Basic Vocabulary Statistics

Model	#Tokens	% Common
qwen-25-15b	151 665	100.00%
quen-25-15b-med	151 665	100.00%
qwen-25-15b-math	151 665	100.00%
Total unique tokens	151 665	—
Tokens common to all models	151 665	100.00%

A.3 Pairwise Vocabulary Overlap

Pair	Common	Only in First	Only in Second
qwen-25-15b vs quen-25-15b-med	151 665 (100.00%)	0 (0.00%)	0 (0.00%)
qwen-25-15b vs qwen-25-15b-math	151 665 (100.00%)	0 (0.00%)	0 (0.00%)
quen-25-15b-med vs qwen-25-15b-math	151 665 (100.00%)	0 (0.00%)	0 (0.00%)

A.4 Token ID Consistency

Common tokens with different IDs: 0 out of 151 665 (0.00%)

A.5 Special Tokens

Model	`unk_token`	`pad_token`	`bos_token`	`eos_token`
qwen-25-15b	None	`<|endoftext|>`	None	`<|endoftext|>`
quen-25-15b-med	None	`<|endoftext|>`	None	`<|im_end|>`
qwen-25-15b-math	None	`<|endoftext|>`	None	`<|endoftext|>`

A.6 Vocabulary Mapping Coverage

- Mapping `quen-25-15b-med` → `qwen-25-15b`: 151 665 / 151 665 (100.00%)
- Mapping `qwen-25-15b-math` → `qwen-25-15b`: 151 665 / 151 665 (100.00%)

A.7 Loaded Tokenizers

- `Llama-3_2-1B-General-lora_model` (bunnycore/Llama-3.2-1B-General-lora_model), vocab size: 128 256
- `llama-3_2-1b-gsm8k-full-finetuning` (axel-datos/Llama-3.2-1B_gsm8k_full-finetuning), vocab size: 128 256
- `Johhny1201/llama3_2_1b_med_QA_2`, vocab size: 128 256

A.8 Basic Vocabulary Statistics

Model	#Tokens	% Common
Llama-3_2-1B-General-lora_model	128 256	100.00%
llama-3_2-1b-gsm8k-full-finetuning	128 256	100.00%
Johhny1201/llama3_2_1b_med_QA_2	128 256	100.00%
Total unique tokens	128 256	—
Tokens common to all models	128 256	100.00%

A.9 Pairwise Vocabulary Overlap

Pair	Common	Only in First	Only in Second
General vs GSM8K-finetune	128 256 (100.00%)	0 (0.00%)	0 (0.00%)
General vs Med-QA	128 256 (100.00%)	0 (0.00%)	0 (0.00%)
GSM8K-finetune vs Med-QA	128 256 (100.00%)	0 (0.00%)	0 (0.00%)

A.10 Token ID Consistency

Common tokens with different IDs: 0 out of 128 256 (0.00%)

A.11 Special Tokens

Model	`unk_token`	`pad_token`	`bos_token`	`eos_token`
General-lora_model	None	`<|finetune_right_pad_id|>`	`<|begin_of_text|>`	`<|eot_id|>`
GSM8K-finetune	None	`<|end_of_text|>`	`<|begin_of_text|>`	`<|end_of_text|>`
Med-QA	None	`<|eot_id|>`	`<|begin_of_text|>`	`<|eot_id|>`

A.12 Vocabulary Mapping Coverage

- Mapping `llama-3_2-1b-gsm8k-full-finetuning` → `Llama-3_2-1B-General-lora_model`: 128 256 / 128 256 (100.00%)
- Mapping `Johhny1201/llama3_2_1b_med_QA_2` → `Llama-3_2-1B-General-lora_model`: 128 256 / 128 256 (100.00%)

B Appendix B: Training details

B.1 Training parameters

Each ensemble model was trained for 5 epochs, each time on the full training dataset, with validation statistics collected at the end of each epoch. Specific parameters of the training are:

- *Entropy weight*: 0.1
- *Dropout % of head's weights:* 10%
- *Head's hidden dimension:* 256
- *Warmup steps:* 50
- *Learning rate:* $1 \cdot 10^{-4}$
- *Random seed:* 42

All trainings were completed on a single A100 40GB NVidia GPU with a batch size of 4, lasting approximately 5 hours per training session. Training was carried out within the Google Colaboratory environment.

B.2 Results: validation losses per epoch

Epoch	Llama (NL)	QWEN (NL)	QWEN (L)
1	1.474	1.049	1.047
2	1.470	1.046	1.043
3	1.469	1.045	1.044
4	1.468	1.045	1.043
5	1.468	1.045	1.045

B.3 Model statistics at evaluation

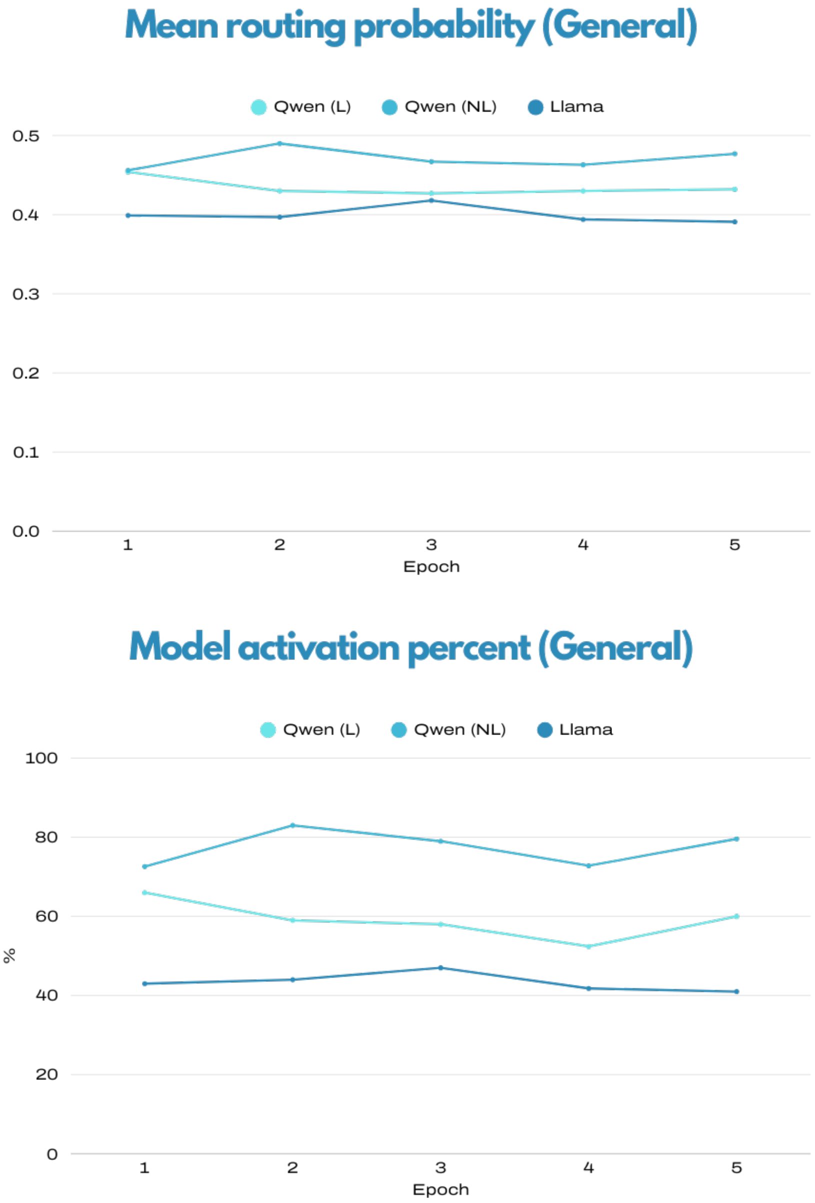


Figure 3: Mean routing probability and activation percent for the General model of each ensemble during validation

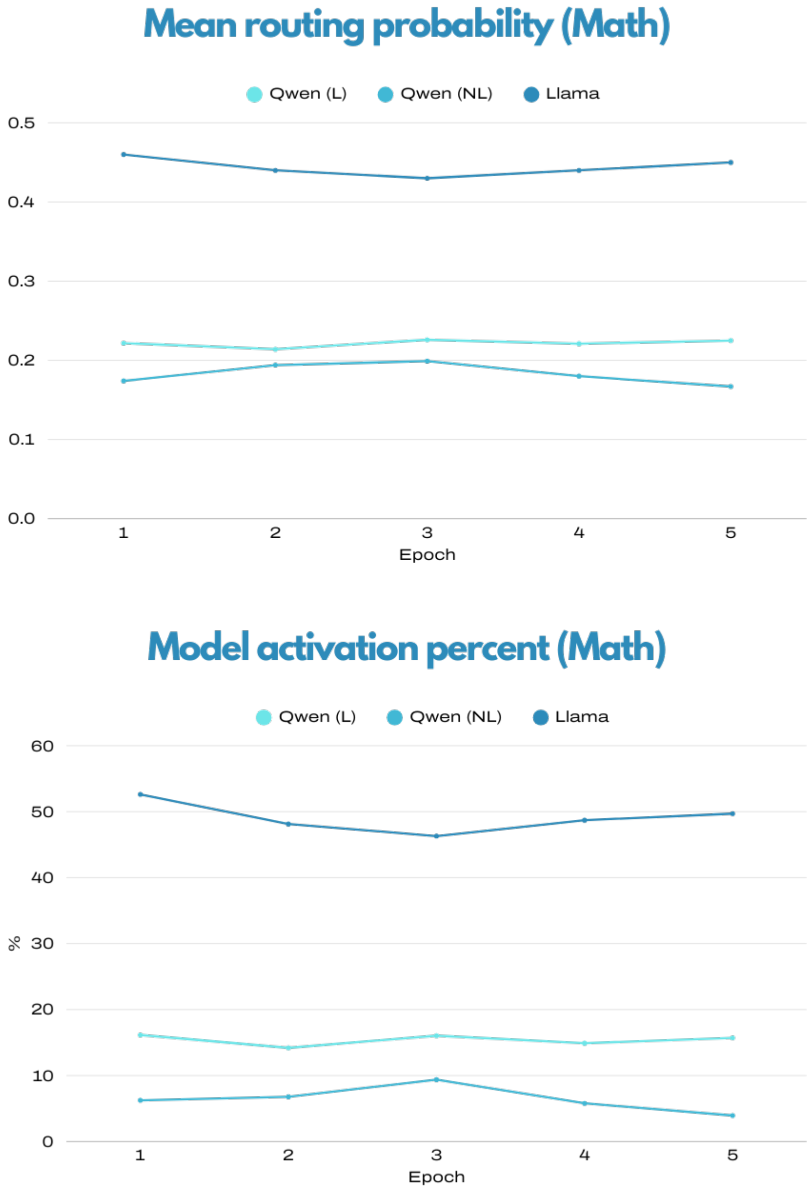


Figure 4: Mean routing probability and activation percent for the Math model of each ensemble during validation

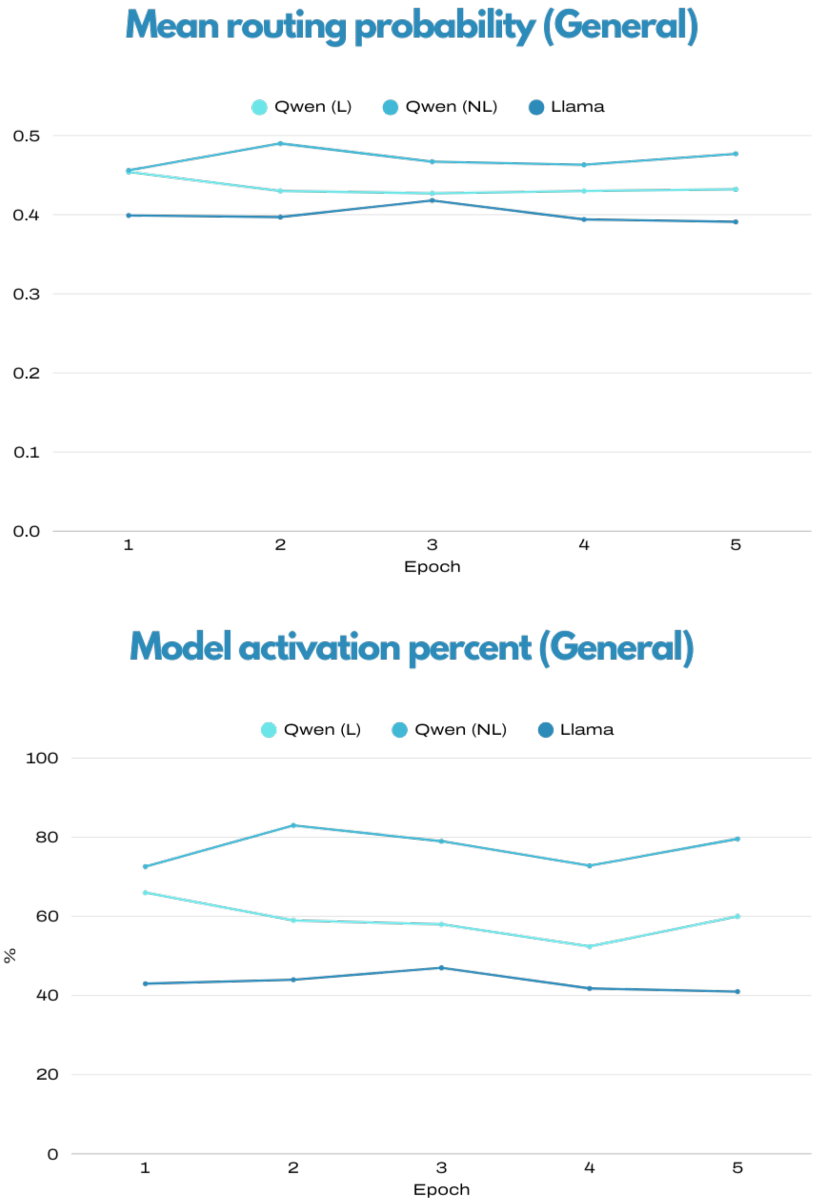


Figure 5: Mean routing probability and activation percent for the Med model of each ensemble during validation

C Appendix C: Evaluation details

C.1 Evaluation parameters

The evaluation process (described in Section 4.6) was carried out on the `test` split of the custom datasets described in Section 3.2. Due to computational constraints, the test dataset was further sampled to 50 examples per task. Each valuation process ran in parallel evaluation for both the saLM-1 model and each of the primitive model, taking approximately 1 hour per task per batch of models. The process was carried out on a A100 4GB NVidia GPU within the Google Colaboratory environment.

C.2 GSM8K Task

For this task, two main statistics were collected for each model:

- *Accuracy*, counting how many times the `math_verify` reported the answer as correct;
- *Average response length.*

Models	Accuracy (%)	Avg. length (chars)
Llama saLM-1	4%	226.1
Llama-3.2-general	28%	152.1
Llama-3.2-med	2%	119.5
Llama-3.2-math	4%	318.4
Qwen saLM-1 (NL)	78%	156
Qwen saLM-1 (L)	80%	165
Qwen-2.5-general	68%	153.3
Qwen-2.5-med	72%	169.3
Qwen-2.5-math	80%	165.3

C.3 MedQA Task

For this task, two main statistics were collected for each model:

- *Accuracy*, counting how many times scraped model answer was the same as the ground truth;
- *Letter detection rate*, counting how many times our script was able to scrape an answer from the model's output.

Models	Accuracy (%)	Detection rate (%)
Llama saLM-1	12%	54%
Llama-3.2-general	22%	74%
Llama-3.2-med	6%	24%
Llama-3.2-math	6%	40%
Qwen saLM-1 (NL)	20%	96%
Qwen saLM-1 (L)	14%	82%
Qwen-2.5-general	18%	100%
Qwen-2.5-med	22%	100%
Qwen-2.5-math	14%	90%