Ensemble Methods

Bhuvan Bedika, Varshith Kada, Abhiram Siripuru, Aneesh Varla
`{bhuvan2023, venkatavks, abhirams, varlaaneesh} @iisc.ac.in`

Abstract

We explore the evolution of ensemble methods, from classical tree-based models to modern sparse expert architectures. Starting with bagging and boosting, we visit Random Forests and XGBoost before turning to Switch Transformers, which apply ensemble ideas within large-scale neural networks. Through controlled experiments on MNIST and WikiText-2, we analyze how these models perform at small scale, reflect on their trade-offs, and evaluate their suitability for different tasks. Our findings highlight how the role of ensembling has shifted — from aggregating weak learners to orchestrating compute across specialized subnetworks.

1 Introduction

Ensemble methods were introduced as a way to build stronger models by combining multiple weaker learners. The idea took shape with early techniques like *Boosting*, first explored by Schapire, and *Bagging*, formalized by Breiman in 1996. Soon after, *AdaBoost* was developed by Freund and Schapire, improving boosting by re-weighting training examples based on previous errors.

Building on these, Breiman introduced *Random Forests* in 2001—a method that fused Bagging with random feature selection. This led to a model that was not only more accurate but also more robust to noise and weak inputs.

In parallel, *Gradient Boosting Machines (GBMs)*, introduced by Friedman (2001), generalized boosting to optimize arbitrary loss functions. This line of work culminated in *XGBoost* (2014), a modern, highly optimized implementation that combined accuracy with computational efficiency through innovations like sparsity-aware split finding, cache-aware access patterns, and out-of-core computation.

As the need for solving more complex tasks grew—especially in areas like NLP, pattern recognition, and multi-domain learning—the focus shifted toward *neural networks* that were not only more powerful but also scalable. A significant milestone in this evolution was the *Mixture of Experts (MoE)* architecture, which activates only a small, specialized subset of the network ("experts") for each input, thereby offering scalability without sacrificing performance. This was further refined with *Switch Transformers*, which simplified the routing mechanism, stabilized training, and significantly reduced computational cost—paving the way for practical deployment of large-scale neural models.

2 Methodology

2.1 Random Forests

We began our exploration of Random Forests with a foundational understanding of decision trees. Breiman's paper on Random Forests [1] was also our first exposure to ensemble methods, through which we learned about bagging, the idea of reducing variance by averaging predictions from trees trained on different bootstrap samples.

Each tree in a Random Forest is trained using the CART methodology, but unlike standard decision trees, they are grown to full depth without pruning. This stood out to us, as pruning is often presented as a primary defense against overfitting in individual trees. We later learned that Breiman's approach instead relies on averaging over a large ensemble of unpruned, randomized trees to control overfitting.

We found the feature-level randomization - selecting a subset of F features at each node (Forest-RI), particularly insightful. It introduced diversity between trees and helped control overfitting in a way we hadn't anticipated. We also explored Forest-RC, which adds further randomness by using linear combinations of features at each split, producing oblique decision boundaries.

Observing how different values of F and model types performed across datasets led us to wonder whether certain models inherently suit specific kinds of data. This inspired a set of experiments (summarized in the appendix, please do check it out) where we tested model structure against data variation, using out-of-bag error estimates to guide tuning.

2.2 XGBoost

XGBoost [2] was our first encounter with boosting methods, and it quickly stood out for both its performance and design. The idea of training trees sequentially on gradients was brilliant. Techniques like sparsity-aware split finding, cache-aware memory access, and support for out-of-core computation made the implementation feel highly optimized for real-world scale.

We found the explicit regularization (via penalties on tree complexity) interesting, in contrast to the randomness-based regularization in Random Forests. We also found its natural handling of missing values, by learning default split directions, to be an elegant engineering solution.

Features like shrinkage and column subsampling further illustrated how carefully the method balances bias, variance, and computational efficiency. Overall, XGBoost left a strong impression not just as a boosting algorithm, but as a well-engineered system.

2.3 Switch Transformers

Switch Transformers [3] marked both our introduction to NLP models and to the Transformer architecture. While the use of a Mixture-of-Experts (MoE) framework, where only a small subset of the model is activated per input, felt conceptually familiar from ensemble methods, the scale and sparsity with which it was applied in this context made it a very different kind of system. It was the first time we saw sparsity used not just as a computational convenience, but as a core design principle for scalability.

The Top-1 routing mechanism, where only the most probable expert processes a given token, initially felt somewhat at odds with the ensemble methods we had studied earlier. While ensembles typically benefit from aggregating multiple perspectives, Switch Transformers instead rely on specialization, activating only a single expert per input. At first, this seemed like a loss of the diversity that ensemble methods leverage, but we came to understand it as a necessary trade-off for scalability. By reducing computation and inter-device communication, this sparsity enables the model to scale to billions of parameters efficiently while maintaining performance.

What caught our attention as we progressed through the paper was the extent to which training such sparse models introduces entirely new kinds of challenges. Dynamic Allocation of Experts meant that experts are not called every time, and the structure of our Transformer is shifing.This cause issues like routing instability due to softmax precision, overfitting in underutilized experts, and weak experts when intialized parameters are large.

Solutions like casting router inputs to float32, applying higher dropout within expert layers, and initializing weights at a smaller scale were brilliant responses to these issues. While they weren't the conceptual core of the model, they helped us appreciate the difference between designing a powerful architecture and making it truly trainable at scale.

Compared to the other methods we studied, Switch Transformers felt like a different class of model, one where engineering constraints and stability considerations weren't afterthoughts, but one of the main issues addressed. The experience broadened our understanding of what building scalable models entails: not just innovation in architecture, but disciplined attention to training dynamics.

A brief explanation for all the models is given in the appendix. This was our personal journey through ensemble methods instead of a generic explanation, this is what we found brilliant and special about these models throughout the history of the field.

3 Experiments

To ground our study in practice, we conducted a set of structured experiments comparing all the models discussed throughout the report — from traditional decision trees to modern gradient-boosted ensembles, and finally to neural network architectures, including sparse expert variants. The goal was to provide a comprehensive comparison of these techniques on a common task and dataset, observing how model design and performance have evolved.

All experiments in this section were conducted on the MNIST digit classification task — a widely studied benchmark that allows for careful comparison of model behavior without overwhelming complexity. Our goal here was to see how each class of model performs when constrained to the same classification setting.

3.1 Decision Trees, Random Forests, and XGBoost on MNIST Classification

We began our empirical study with traditional tree-based models to understand their evolution and limitations in a modern classification setting. All models were evaluated on the MNIST dataset using accuracy, precision, recall, and F1 score.

Experimental Setup. Each model was implemented following the methods described in its respective foundational paper. We tested a CART-based decision tree, Random Forests using the Forest-RI variant with different values of F, and XGBoost using its default boosting framework.

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Decision Tree	87.24%	87.28%	87.24%	87.24%
Forest-RI ($F = 1$)	93.77%	93.77%	93.50%	93.60%
Forest-RI ($F = 10$)	94.33%	94.33%	94.11%	94.16%
XGBoost	97.95%	97.95%	97.95%	97.95%

Table 1: Performance of tree-based models on MNIST classification.

Interpretation. As seen in Table 1, XGBoost offers the best performance among tree-based models, significantly improving on the basic decision tree through boosting and regularization. Random Forests also provide strong results, particularly with larger F, benefiting from reduced correlation among trees due to randomized feature selection.

3.2 Switch MLP vs Dense MLP on MNIST Classification

Model Setup. We constructed a baseline *Dense MLP* with three hidden layers, each using FLOP-matched dimensions relative to the Switch models to ensure fair comparison. We then modified this architecture by replacing the middle hidden layer with a Switch layer. Each expert in the Switch layer is a two-layer feed-forward network with ReLU activation and dropout. We created five Switch MLP variants with $N = \{2, 4, 6, 8, 10\}$ experts.

All stabilizing techniques discussed in the original Switch Transformer paper [3] were implemented: selective precision casting for routing logits, expert dropout, and smaller initialization. Each model was trained for 100 epochs on the MNIST training set and evaluated periodically on the test set for loss and accuracy.

Results. Figure 1 plots test accuracy across epochs. Surprisingly, the dense MLP consistently outperforms all Switch variants, achieving higher accuracy and converging faster. This trend is mirrored in the test loss curves and time-per-epoch plots provided in Appendix C. The Switch MLPs show slower convergence and slightly higher loss at every evaluation point, despite using the same number of active parameters per input.

Final Test Metrics:

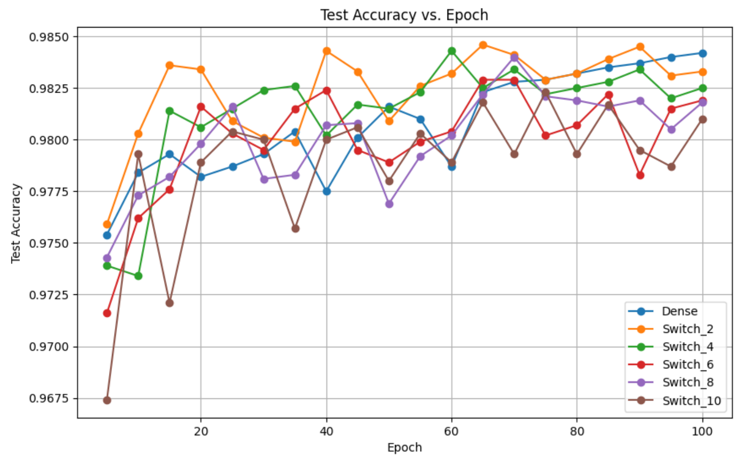

Figure 1: Test Accuracy vs. Epoch for Dense and Switch MLP variants.

Model	Accuracy	Precision	Recall
Dense (Standard MLP)	0.9842	0.9841	0.9841
Switch MLP (10 Experts)	0.9810	0.9808	0.9809

We also tested a Switch MLP with 18 experts. While the model remained stable, it did not yield improved performance over the 10-expert version and incurred significant additional training time. The corresponding performance and runtime curves are shown in Appendix C.

Interpretation. The weaker performance of Switch MLPs on MNIST can be attributed to the simplicity of the task. MNIST classification is linearly separable to a large extent, and dense MLPs already perform very well with minimal overhead. The added complexity of expert routing and dropout in sparse models introduces instability and slows convergence without offering meaningful gains.

Unlike in language modeling, where each token introduces more variation and depth of structure, classification tasks like MNIST offer limited opportunity for expert specialization. In such scenarios, the overhead of sparse activation outweighs its benefits. As confirmed in our NLP experiments (Appendix B), Switch models are far more effective in sequence modeling tasks, where token diversity and data scale allow expert layers to specialize meaningfully.

While it is possible that Switch models would catch up with more training, such a strategy undermines their intended advantage: efficiency. In this setting, the dense MLP is not only faster and simpler but also delivers stronger performance — making it the better choice for low-scale classification.

Final Conclusion of the Experiment While tree models perform well, particularly XGBoost, they are still outperformed by both dense MLPs and switch MLPs trained on the same task. Switch MLPs did not offer improvements to dense MLPs in this setting, highlighting that for simple, low-dimensional classification tasks, the overhead introduced by sparse routing outweighs its benefits. Nonetheless, the progression across these models — from decision trees to ensemble methods and then to neural networks — demonstrates clear improvements in accuracy, learning capacity, and scalability.

Conclusion

Our study traced the arc of ensemble modeling from decision trees to sparse Transformers. Each model class introduced new ways of reducing error — through bagging, boosting, routing, or regularization. While tree ensembles like XGBoost perform well on simple tasks, dense MLPs outshine them with enough scale. Switch models, though less suited for low-dimensional classification, showed clear advantages in NLP settings. Ultimately, the choice of model depends not just on accuracy, but on task complexity, scalability, and where the model's inductive bias aligns best.

A Detailed Methodology Descriptions

A.1 Switch Transformer Architecture and Training Details

A.1.1 Architecture and Routing Mechanism

Switch Transformers [3] implement a sparse Mixture-of-Experts (MoE) design within select Transformer layers, enabling substantial model scaling without linear growth in computational cost. Each Switch layer replaces the standard feed-forward network (FFN) with a router and a bank of N expert networks — independently parameterized FFNs.

Given a token with hidden representation x, the router computes scores $h(x)_i = W_r \cdot x$ for each expert i, and applies a softmax over these scores to produce the routing distribution:

$$p_i(x) = \frac{e^{h(x)_i}}{\sum_{j=1}^{N} e^{h(x)_j}}.1 \tag{1}$$

Each token is then routed to the single expert with the highest $p_i(x)$ (Top-1 routing). This sparse activation scheme reduces both memory and computation, avoiding the overhead of aggregating multiple expert outputs while maintaining model quality. The result is a model with a large number of parameters, but where each token only activates a small fraction of the network during inference or training.

A.1.2 Load Balancing Auxiliary Loss

Top-1 routing introduces the risk of expert overload and underutilization. To address this, a load balancing auxiliary loss is added to the training objective. For each batch $\mathcal{B}$ of T tokens, let f_i denote the fraction of tokens routed to expert i, and P_i the average routing probability assigned to expert i:

$$f_i = \frac{1}{T} \sum_{x \in \mathcal{B}} \nVdash \left\{ \arg\max_j p_j(x) = i \right\}, \quad P_i = \frac{1}{T} \sum_{x \in \mathcal{B}} p_i(x). \tag{2}$$

The loss is computed as:

$$\mathcal{L}_{balance} = \alpha \cdot N \cdot \sum_{i=1}^{N} f_i \cdot P_i, 2 \tag{3}$$

where α is a small constant (e.g., 10^{-2}). This auxiliary term penalizes configurations where the routing probability mass and the actual usage of experts are misaligned, encouraging more even expert activation and improving overall capacity utilization.

A.1.3 Training Stability and Regularization Techniques

Training instability in Switch Transformers arises from their sparse activation pattern and dynamic routing behavior. Since only one expert is activated per token, each token effectively sees a different sub-network at each step. Moreover, each expert is exposed to a much smaller portion of the training data, receiving fewer gradient updates than layers in dense models. This leads to instability and overfitting risks that must be addressed through architectural and system-level adjustments.

Selective Precision for the Router While most of the model operates in reduced precision (e.g., bfloat16 or float16) for performance efficiency, the routing computation is particularly sensitive to numerical instability. In contrast to attention mechanisms — where softmax is used for weighted aggregation — the router's softmax output is used to make a discrete Top-1 selection. Small rounding errors in logits due to low precision can lead to incorrect expert assignment.

To mitigate this, the token representation x is temporarily cast to float32 before computing the routing logits $h(x)_i$ and softmax probabilities. This ensures accurate expert selection. After routing, all computations return to the lower precision. This approach provides the stability benefits of float32 in the critical routing step without incurring significant computational or memory overhead across the rest of the model.

Expert Dropout Regularization Due to their sparse activation, experts are updated on only a small fraction of training tokens. This limited exposure increases the risk of overfitting — particularly in fine-tuning scenarios with fewer labeled examples. To address this, Switch Transformers employ increased dropout rates inside the expert FFNs. This "expert dropout" reduces co-adaptation within individual experts and serves as a form of stochastic regularization. It is conceptually similar to ensemble regularization: each expert receives different gradient flows depending on token routing and dropout masking, promoting more robust and generalizable behavior.

Smaller Weight Initialization Another stabilization strategy involves reducing the initialization scale of weights in the expert and router layers. The standard Transformer initialization uses a truncated normal distribution with standard deviation $\sigma = \sqrt{s/n}$, where n is the fan-in and $s = 1.0$. In Switch Transformers, s is reduced to 0.1, resulting in smaller initial weights.

This adjustment addresses a key issue: early in training, experts with higher initial outputs may dominate routing decisions via the softmax, leading to uneven expert usage and potential collapse in utilization diversity. Since each expert is already exposed to fewer updates due to sparse activation, such early imbalances can persist and hinder convergence. Smaller initializations ensure more uniform routing probabilities and reduce variance in early training, improving both stability and repeatability across runs.

A.1.4 Efficiency and Performance Benefits

By activating only one expert per token, Switch Transformers reduce computation per token to levels comparable with dense models, while scaling the parameter count significantly. This leads to favorable compute-to-accuracy trade-offs. Empirical results show that Switch Transformers outperform dense Transformers and traditional MoEs when matched on FLOPs and continue to perform competitively even when dense models are given more compute.

These gains are made possible not just by sparse routing, but by careful attention to system-level training dynamics. Precision sensitivity, expert-specific regularization, and tailored initialization are all critical to ensuring that the model's theoretical benefits translate into practical performance at scale.

A.2 Random Forests

Random Forests [1] are an ensemble learning method that combines bagging with feature-level randomization to reduce variance and improve generalization. Each individual decision tree in the ensemble is trained on a bootstrap sample, which is a subset of the training data sampled with replacement. This foundational technique, known as bagging, stabilizes the model by reducing overfitting through averaging uncorrelated predictors.

Random Forests further introduce randomness at the level of feature selection. In the **Forest-RI** (Random Input) variant, each split in a tree considers only a random subset of F features rather than evaluating all available features. This feature-level randomness increases tree diversity by ensuring that different trees explore different parts of the feature space, even when trained on similar data. The resulting ensemble is less correlated and more robust, particularly in noisy or weak-signal datasets.

The optimal value of F can be selected using out-of-bag (OOB) error estimates. Since each tree is trained on a different subset of data, approximately one-third of the data is left out per tree. This OOB data serves as a built-in validation set, allowing performance estimation and hyperparameter tuning without requiring a separate holdout set.

The **Forest-RC** (Random Combination) variant introduces further diversity by using oblique splits. At each node, L features are chosen at random, and F random linear combinations of these features are computed. The best split is selected from among these constructed features. This leads to more complex, non-axis-aligned decision boundaries and increases the representational capacity of individual trees.

A key property of Random Forests is that their robustness comes not from controlling the complexity of individual models, but from the diversity of the ensemble. Overfitting is mitigated by averaging over many weakly correlated trees, each trained on different data and feature subsets. This makes Random Forests effective when input data is noisy, high-dimensional, or contains irrelevant features.

In comparison with boosting-based methods such as AdaBoost, Random Forests tend to perform more stably on datasets with noise or label irregularities. The regularization effect comes not from penalizing model complexity, but from injecting randomness, a principle that is simple and effective across a wide range of domains.

A.3 XGBoost

XGBoost [2] is a scalable and highly optimized implementation of gradient boosting. It builds an ensemble of decision trees in a sequential manner, where each new tree is trained to correct the residual errors of the current model. Specifically, the objective function is minimized by fitting each new learner to the gradient of the loss with respect to the previous prediction. This gradient-based optimization allows the ensemble to make progress along the steepest descent direction in function space, improving accuracy at each step.

A distinguishing feature of XGBoost is its explicit incorporation of regularization. The training objective includes penalties on both the number of leaves and the magnitude of leaf weights, discouraging overly complex trees. This regularization addresses overfitting directly, which is particularly important in boosting algorithms, where each new tree tends to focus increasingly on harder (and potentially noisier) examples.

In contrast to Random Forests, where diversity is introduced through sampling and feature selection, XGBoost controls overfitting by shaping the optimization landscape through explicit regularization. These two approaches are fundamentally different, and each offers distinct advantages depending on the characteristics of the dataset.

XGBoost also introduces several system-level innovations that contribute to its performance and scalability:

- The tree construction algorithm sorts feature values once and reuses this structure across nodes, significantly reducing computation.
- Unlike greedy tree learners that stop growing based on impurity thresholds, XGBoost grows full trees and prunes them depth-wise using a maximum gain threshold.
- During training, the algorithm natively handles missing or sparse data by learning default directions for split decisions, avoiding the need for imputation or preprocessing.

To further enhance generalization, XGBoost applies **shrinkage**, a technique where the contribution of each new tree is scaled by a learning rate η. This slows down learning and reduces the influence of any single tree, improving robustness. Additionally, **column subsampling**, in which subsets of features are randomly selected at each iteration, is used to inject diversity and reduce variance. This technique, similar in spirit to that used in Random Forests, is adapted here for the boosting framework.

Together, these features make XGBoost an effective tool for structured prediction tasks. Its performance stems not only from the boosting framework, but also from the integration of regularization, optimization, and efficient system design, resulting in fast, stable, and accurate learning across a wide range of datasets.

B NLP on Switch Transformer at low scale

B.1 Comparing Dense and Switch Transformer Variants

In an earlier experiment, we observed that a dense MLP outperformed its Switch counterpart in a classification task. This was surprising given the architectural benefits of sparse models, but we attributed it to the simplicity of the task and model size: for small, low-dimensional networks, the overhead of routing in Switch layers outweighed the benefits. This raised a question we wanted to explore more broadly: *How do Switch Transformers perform on NLP tasks at a similar scale?*

Objective. We designed this experiment to directly compare the behavior of Switch and dense Transformers in low-scale settings for language modeling. Specifically, we wanted to investigate whether the same degradation seen in Switch-MLPs applies to small Transformer-based models on NLP tasks, or whether Switch still offers gains even at these sizes.

Model and Setup. We trained a compact 4-layer Transformer with model dimension $d_{model} = 128$, feedforward size $d_{ff} = 256$, and 4 attention heads. This was our dense baseline. For the Switch Transformer variants, we replaced every alternate feed-forward layer with a Switch Layer using N experts (for $N \in \{2, 4, 6, 8, 10\}$). Each expert was a 2-layer MLP. As in standard Switch Transformers [3], each token was routed to a single expert based on a learned softmax over scores $h(x) = W_r \cdot x$.

Models were trained for 10,000 steps on the WikiText-2 dataset for next-token prediction. All hyperparameters (learning rate, batch size, etc.) were shared across models. Importantly, despite the y-axis label reading "negative log perplexity," we report $\log(perplexity)$ throughout for clearer interpretability.

Results. Figure 2 shows a zoomed-in section of the training curves centered around step 3000. Here, the Switch variants, particularly with 10 experts, begin to slightly outperform the dense baseline in terms of log-perplexity. This supports the claim that sparse expert routing begins to confer advantages even at low scale, at least temporarily.

However, as training continues, these differences gradually diminish. By around step 5000 and beyond, all models - regardless of the number of experts - converge to nearly identical performance levels, as seen in Figure 3. The full training curve in Figure 4 confirms this behavior.

We believe this convergence is likely due to diminishing returns from continued training: the dataset is relatively small, the model is underparameterized by modern standards, and the early gains from routing specialization may be absorbed by overfitting or saturation of representational capacity. In other words, once the model reaches a certain level of fit, architectural differences matter less than the limited data or optimization headroom available.

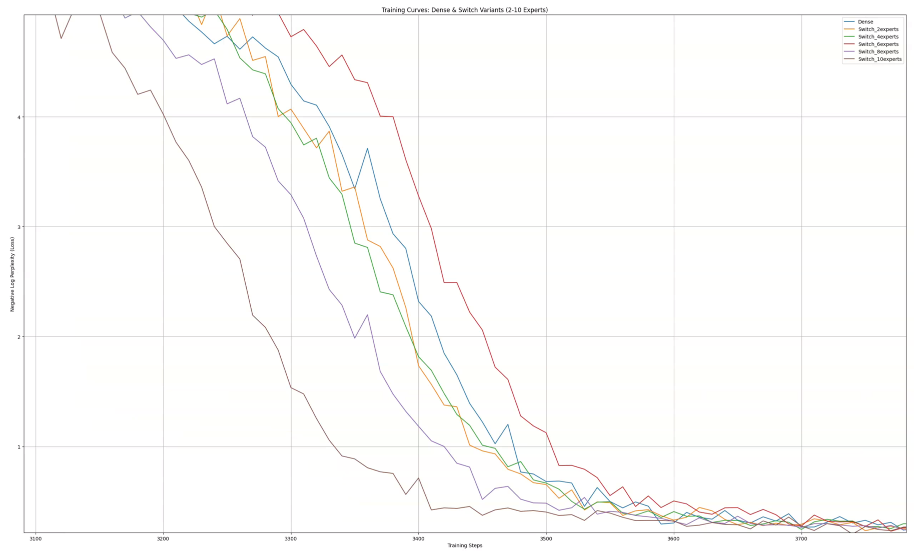

Figure 2: Performance around step 3000. Switch Transformers begin outperforming dense baseline.

Interpretation. While Switch Transformer variants achieve slightly better log-perplexity around step 3000, all models ultimately converge to comparable performance. This convergence may be attributed to one or more factors: limited model size reducing potential for divergence, the saturation of learning due to dataset simplicity, or early overfitting balancing out across architectures.

Nonetheless, the experiment supports our core hypothesis. Unlike our earlier classification experiments, where sparse MLPs underperformed due to routing overhead, Switch Transformers maintain parity and even temporary advantage over dense models in language modeling. These results suggest

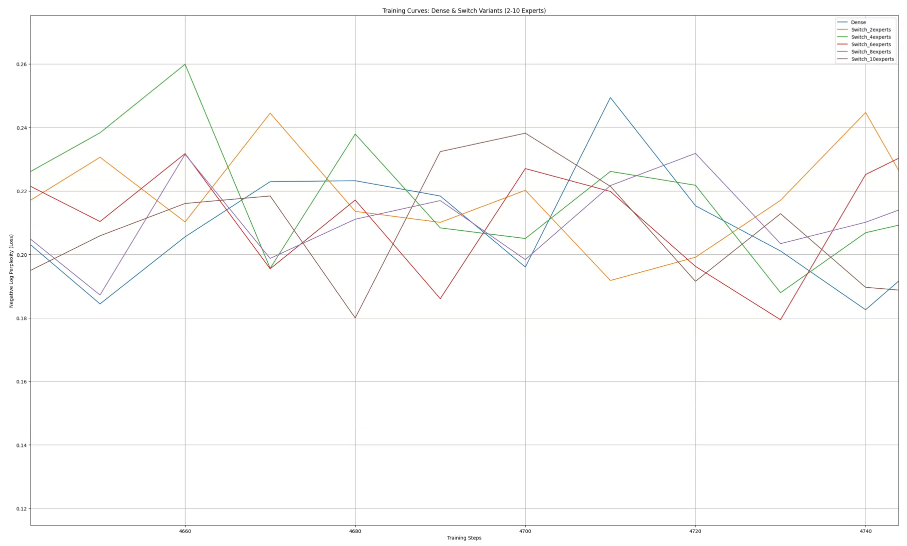

Figure 3: Post-convergence snapshot. All models reach similar log-perplexity after around 4000 steps.

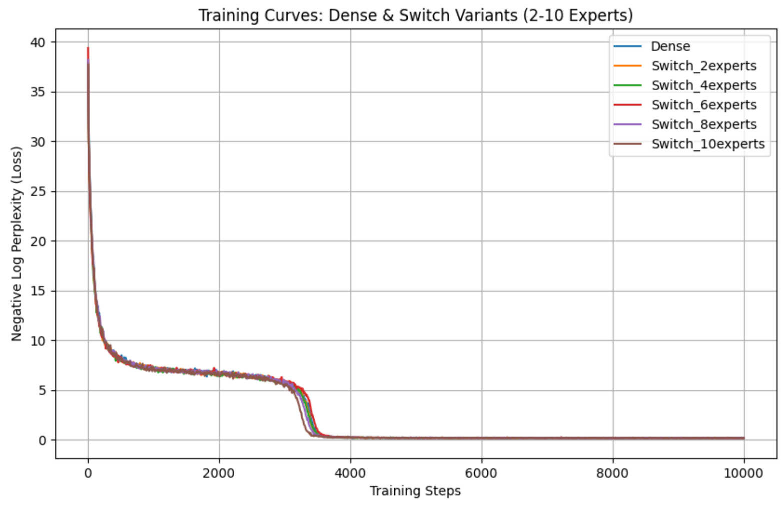

Figure 4: Full training curves. Long-term convergence across all model variants.

that the benefits of sparsity manifest more naturally in NLP settings, even at small scale. The trend also aligns with existing literature that shows the full value of sparse MoE architectures becomes increasingly apparent as models are scaled up in width and depth [3]. Our experiment confirms that Switch models are more naturally suited for NLP than for simple classification, particularly when training dynamics, token diversity, and model depth allow experts to specialize meaningfully.

In summary, this experiment supported our hypothesis: at small scale, Switch Transformers can still outperform dense models for language modeling, a result not mirrored in our classification trials. Given that the benefits of Switch models increase with scale, this provides strong motivation to explore their application in more complex NLP tasks.

B.2 Distillation from Switch Transformer Teacher

Having established that Switch Transformers offer competitive performance to dense Transformers even in low-scale language modeling tasks, we next explored whether this performance can be effectively transferred to a smaller model through knowledge distillation at lower scales.

Objective. Our goal was to examine whether a compact dense Transformer can match the performance of a larger Switch-based teacher model via distillation (at lower scales). Specifically, we used the best-performing Switch-10 Experts model from the previous section as the teacher and trained a dense Transformer with the same architecture as the baseline to mimic its output distribution.

Distillation Method. The distillation process followed standard techniques: rather than training on ground truth labels, the student model was trained to minimize the Kullback-Leibler (KL) divergence between its predicted token probabilities and those of the teacher. The loss function is defined as:

$$\mathcal{L}_{KD} = \sum_{x} KL(Softmax(z^{teacher}(x)) \parallel Softmax(z^{student}(x))) \tag{4}$$

where $z^{teacher}(x)$ and $z^{student}(x)$ are the pre-softmax logits of the teacher and student models for token x.

Results. The training was run for 10,000 steps, and log-perplexity values were recorded throughout. Figure 5 shows the full training curves for the teacher (Switch-10 Experts), the student (Distilled Dense Transformer), and the dense baseline. A zoomed-in view is provided in Figure 6.

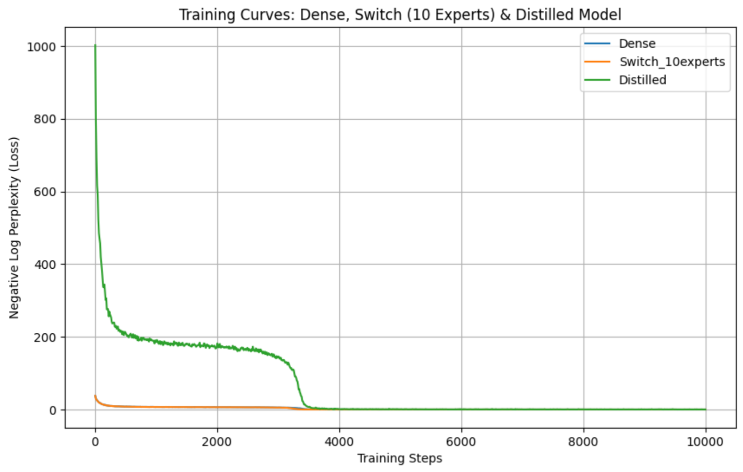

Figure 5: Training Curves for Dense, Switch-10 (Teacher), and Distilled Transformer.

By approximately step 3500, the loss of the distilled model converges closely with both the dense and teacher models. This alignment holds through the remainder of training. Final log-perplexity values are as follows:

Model	Final Log-Perplexity Loss
Dense Transformer	0.2018
Switch-10 Experts (Teacher)	0.2026
Distilled Student	0.2043

The distilled model achieves a final loss that is only marginally higher than both its teacher and the dense baseline. Although the original Switch Transformer paper reported that distillation retained about 30% of the accuracy improvement, in our case, the teacher model had only a minor advantage. Calculating the gain retention:

$$GainRetention = \frac{0.2043 - 0.2018}{0.2026 - 0.2018} \approx 0.3125$$

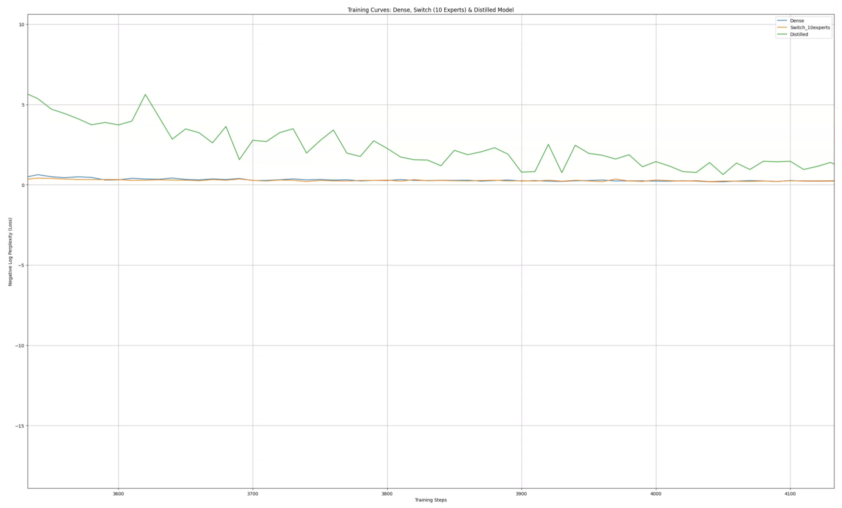

Figure 6: Zoomed-in: Convergence of models around step 3500.

This implies the student recovered approximately 31% of the teacher's small gain over the baseline, consistent with the findings of prior work despite the narrower performance gap.

Conclusion. Distillation remains a powerful method for compressing knowledge from Switch-based models, even when the performance gap is narrow. The student model trained using this technique effectively matches the teacher's behavior, confirming the viability of Switch Transformers not only as high-capacity learners but also as effective knowledge sources for smaller models.

C Investigating the Effectiveness of $F = 1$ in Random Forests

In Briemans Random Forest Paper [1], we noticed that using $F = 1$, i.e., choosing a single random feature at each split yielded unexpectedly strong performance, comparable to much higher values of F. This was surprising, as intuitively, selecting splits based on just one feature per node should be more susceptible to weak splits and instability.

To understand this phenomenon, we explored two hypotheses:

1. High Feature Strength (Low Redundancy). Perhaps all features were individually strong, so any randomly chosen feature would result in a good split. To test this, we constructed a dataset with high feature redundancy (15 out of 20 features redundant) using `make_classification`. We evaluated Random Forests with both $F = 1$ and $F = \log_2 M + 1$. Despite redundancy, the $F = 1$ model continued to perform well, suggesting that the strong performance was not solely due to uniformly strong features.

2. High Feature Correlation. Alternatively, perhaps the features were highly correlated — so any of them would behave similarly when used to split the data. We created a synthetic dataset with 5 informative features and 15 highly correlated ones by linearly combining features with small noise. Again, both $F = 1$ and $F = \log_2 M + 1$ were evaluated. While performance dropped slightly, $F = 1$ still remained competitive, indicating that correlation alone was not the main cause.

Conclusion. These results suggest that the effectiveness of $F = 1$ is not simply due to data bias or redundancy. Instead, it appears to reflect a fundamental strength of the Random Forest algorithm: the extreme randomness introduced at each node (via $F = 1$) leads to lower correlation between trees, which in turn improves ensemble diversity. As discussed in Breiman's original paper [1], reducing correlation — even at the cost of individual tree strength — can improve overall generalization when averaging across many trees.

Results Summary.

Experiment	**Accuracy**	**Strength**	**Correlation**	c/s^2
High Redundancy, $F = 1$	94.75%	0.9534	0.1613	0.1774
High Redundancy, $F = \log_2 M + 1$	95.33%	0.9582	0.1613	0.1750
High Correlation, $F = 1$	92.60%	0.9233	0.1844	0.2166
High Correlation, $F = \log_2 M + 1$	92.96%	0.9258	0.1882	0.2195

Table 2: Performance and internal metrics from redundancy/correlation experiments.

D Results that were not included

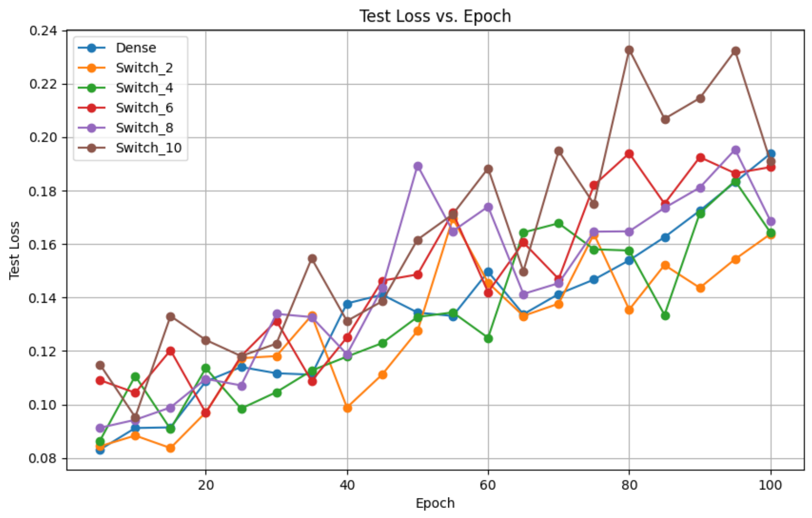

Figure 7: Test Loss vs. Epoch. Dense MLP achieves lower loss across training.

E Individual Contributions

Bhuvan Bedika: XGBoost Reading, Experiment Design, Report **Varshith Kada**: Switch Transformers Reading and Experiment design, Random Forests Reading and Experiment design, Report **Abhiram Siripuru**: Random Forests Reading and Experiment design, Experiment and Code Running, Presentation **Aneesh Varla**: XGBoost Reading, Experiment Design, XGBoost Experiment and code Running, Presentation

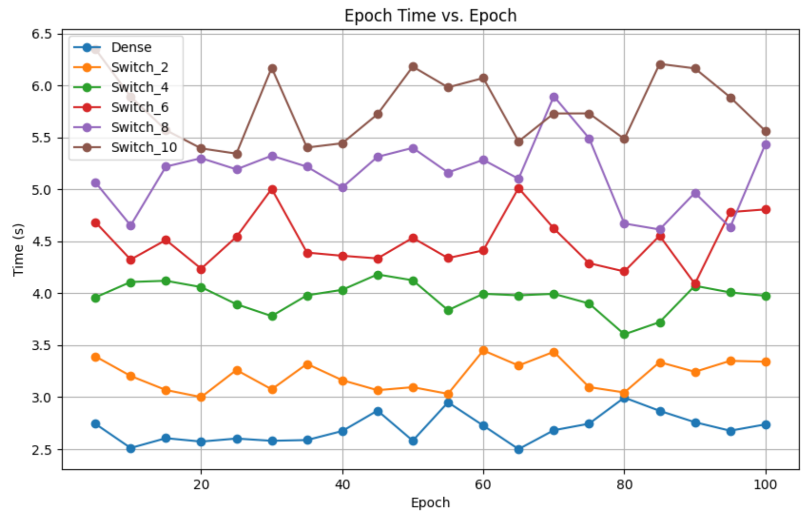

Figure 8: Training Time per Epoch. Switch MLPs become more expensive as expert count increases.

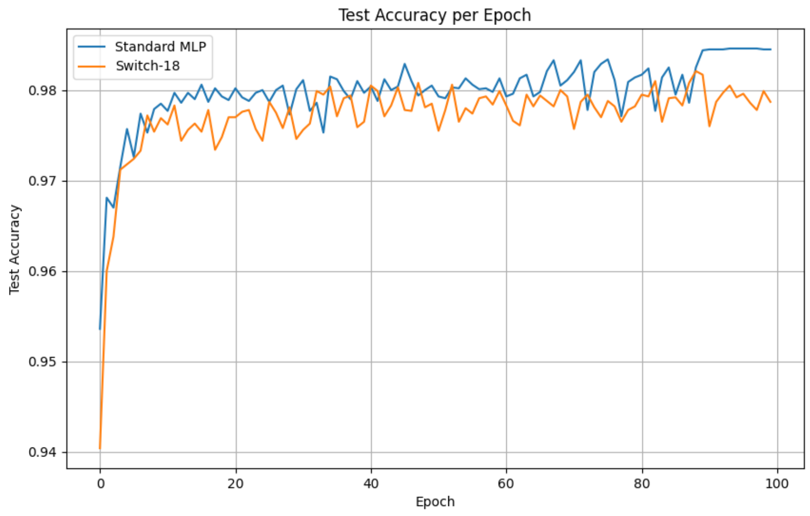

Figure 9: Test Accuracy vs. Epoch for Switch MLP with 18 experts.

References

[1] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[2] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016.

[3] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022.