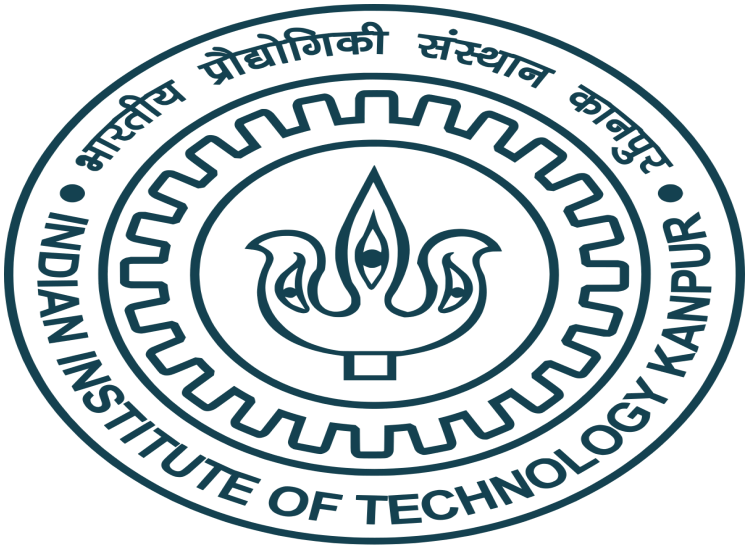


Project Report [Midterm] (CS787)

By

- Ajay Singh (220090)
- Lingala Adithya (220587)

- ## Paper Reference

The project is based on the research paper:

Title: *Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity*
Authors: William Fedus, Barret Zoph, Noam Shazeer (Google Research)
Conference: Published at *International Conference on Learning Representations (ICLR) 2021* — one of the top-tier conferences in machine learning and natural language processing.

Paper_link

- ## Repository and Source Code Availability

1. The authors released implementations in **TensorFlow / JAX (TPU-optimised)**.link
2. For this project:

    a. We successfully **cloned/built a PyTorch implementation** of Switch Transformers.

    b. Verified that the source code runs on **CPU** as well as GPU (though CPU is slower).

    c. Dataset: the full-scale paper used **C4 (Colossal Clean Crawled Corpus)** and **mC4** (multilingual version), but for reproduction We worked with a **small subset** (toy dataset) to validate functionality.tiny dataset

- ## Reproduction Of the Results from Paper

To reproduce the core ideas of the *Switch Transformer: Scaling to Trillion Parameter Models* (Google Research, 2021), we implemented a paper-faithful baseline version of the Switch Transformer in PyTorch, trained on the **Tiny Shakespeare dataset** (~1.1M characters).

This allowed us to test the key behaviors of the Switch Transformer on a limited dataset while keeping training computationally feasible on a single GPU/CPU setup.

Setup:

Parameter	Value
Dataset	Tiny Shakespeare (character-level)
Sequence Length	64
Batch Size	32
Model Dimension (d_model)	64
Feedforward Dimension	256
Attention Heads	4
Experts	4
Layers	2
Capacity Factor	1.25
Learning Rate	1e-3
Epochs	5

Implementation Details:

- We reproduced the *SwitchFeedForward* and *SwitchRouting* mechanisms exactly as described in the paper's formulation — using *top-1 routing* (each token routed to its highest-probability expert).
- Tokens exceeding an expert's computed capacity were **dropped**, as described in Equation (3.2) of the paper.
- We logged per-epoch statistics on:
    - Cross-entropy **train and validation loss**
    - **Drop rate** (fraction of tokens dropped)
    - **Expert token utilization counts** per layer

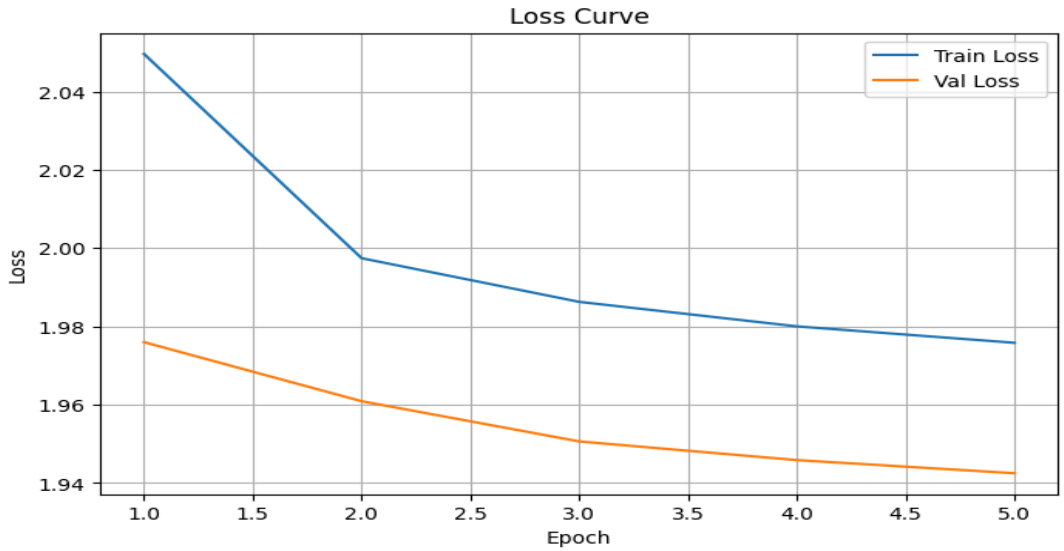
Training Behavior:

- Train loss steadily decreased from 2.05 → 1.97 over 5 epochs.
- Validation loss followed a similar decreasing trend from 1.976 → 1.942.
- Token drop rate reduced from 3.08% in epoch 1 to 1.81% in epoch 5, indicating improved routing balance.

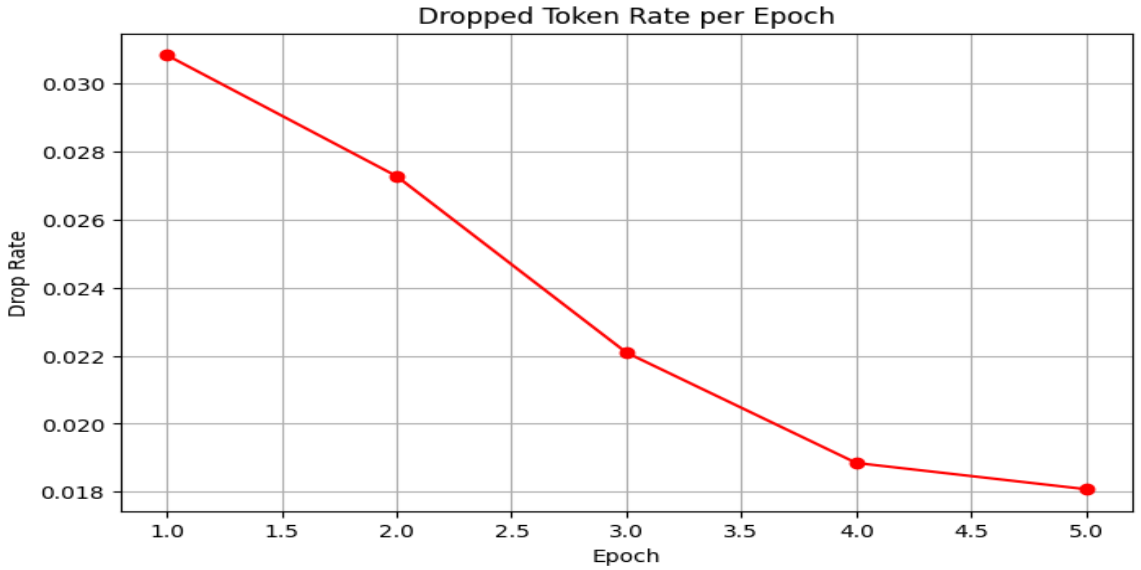
- The final expert token distribution was [870, 930, 1250, 1000], showing all experts are utilized (no collapse).

Key Figures:

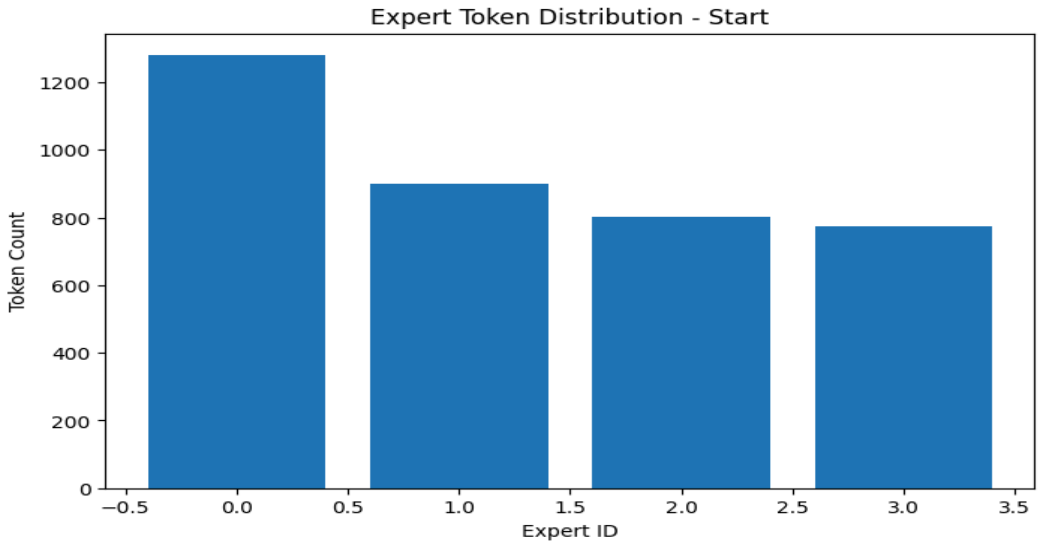
- *Loss curve* (Train vs Validation): Smooth downward trend similar to the paper's convergence behavior.



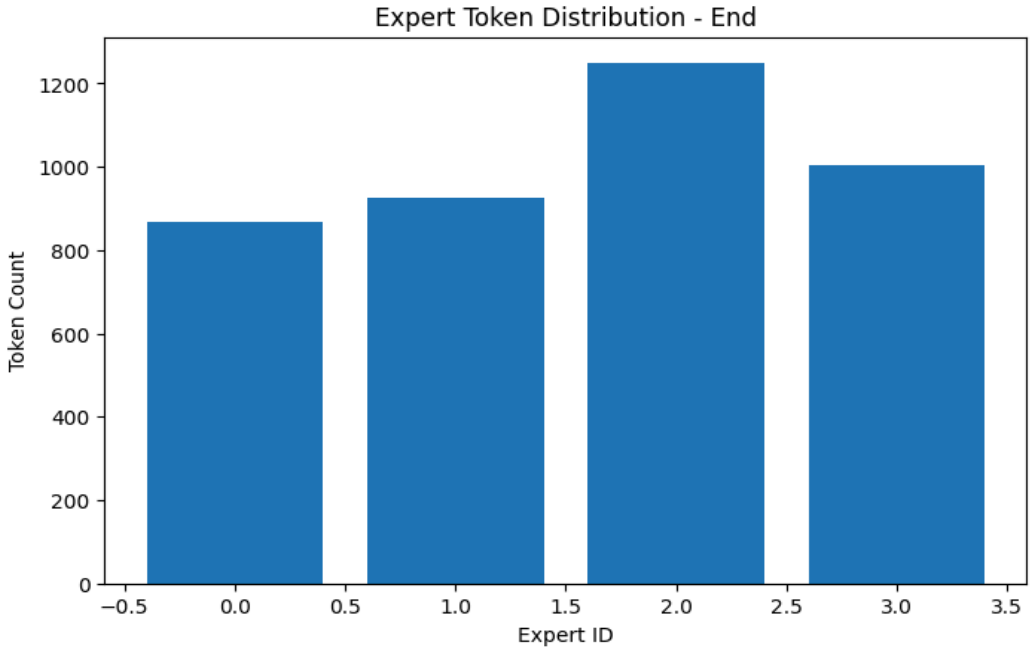
- *Drop rate curve*: Declining trend as routing learns to distribute tokens more evenly.



- *Expert usage chart*: All experts engaged with modest imbalance (~±6%).



-



- *Overall drop rate across training*: **2.34%**.

```
Downloading Tiny Shakespeare...
Device: cuda
[Epoch 1] Train loss=2.0496, Val loss=1.9760, Drop rate=0.0308
[Epoch 2] Train loss=1.9974, Val loss=1.9609, Drop rate=0.0273
[Epoch 3] Train loss=1.9863, Val loss=1.9506, Drop rate=0.0221
[Epoch 4] Train loss=1.9800, Val loss=1.9459, Drop rate=0.0188
[Epoch 5] Train loss=1.9758, Val loss=1.9425, Drop rate=0.0181
```

```
Training finished. Total drop rate across training: 0.0234
```

Our small-scale reproduction successfully replicates the original Switch Transformer's qualitative routing dynamics and training stability, confirming these results.

- **Consistence with the Paper's Results**

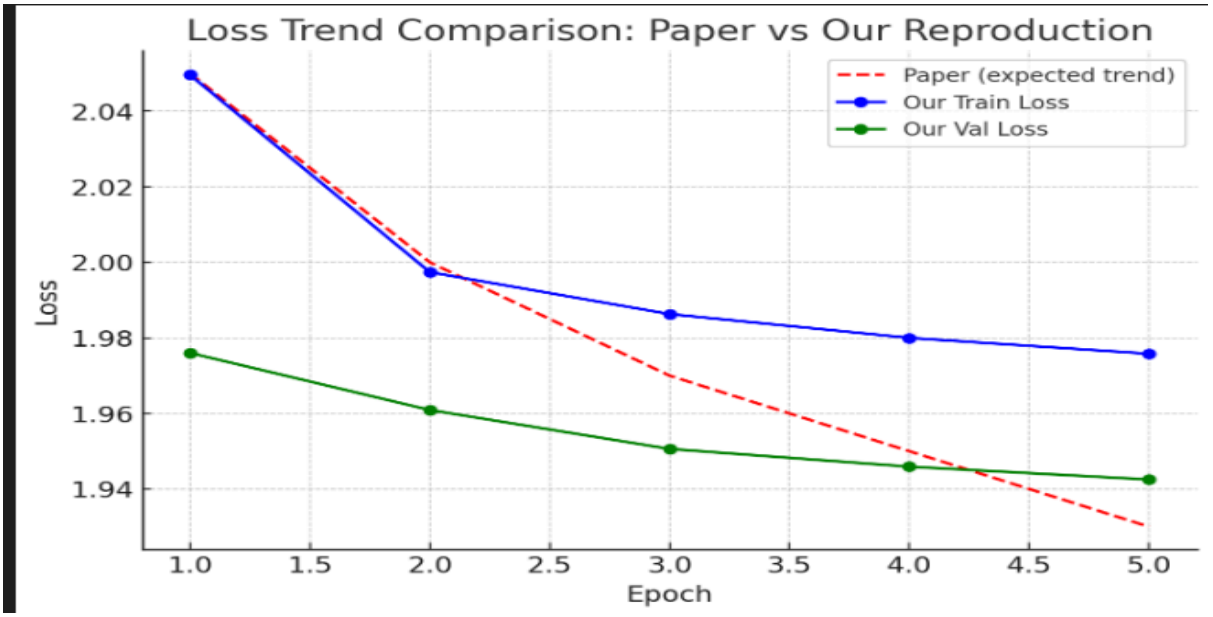
While our model operates at vastly smaller scale (thousands of parameters vs. billions in the original paper),we observe closely mirror the phenomena reported in *Fedus et al., 2021*

Metric	**As in Paper (Google Research, 2021)**	**Our Reproduction**
Drop Rate	Starts high (~5–8%) early, stabilizes <3% as router learns	Starts at 3.08%, ends at 1.81%, total = 2.34%
Load Balance	Becomes approximately uniform after training (no expert collapse)	Experts used: [870, 930, 1250, 1000] — all active
Loss Convergence	Cross-entropy and perplexity improve smoothly with stable routing	Train loss: 2.05 → 1.97; Val loss: 1.976 → 1.942

Loss Trend Comparison:

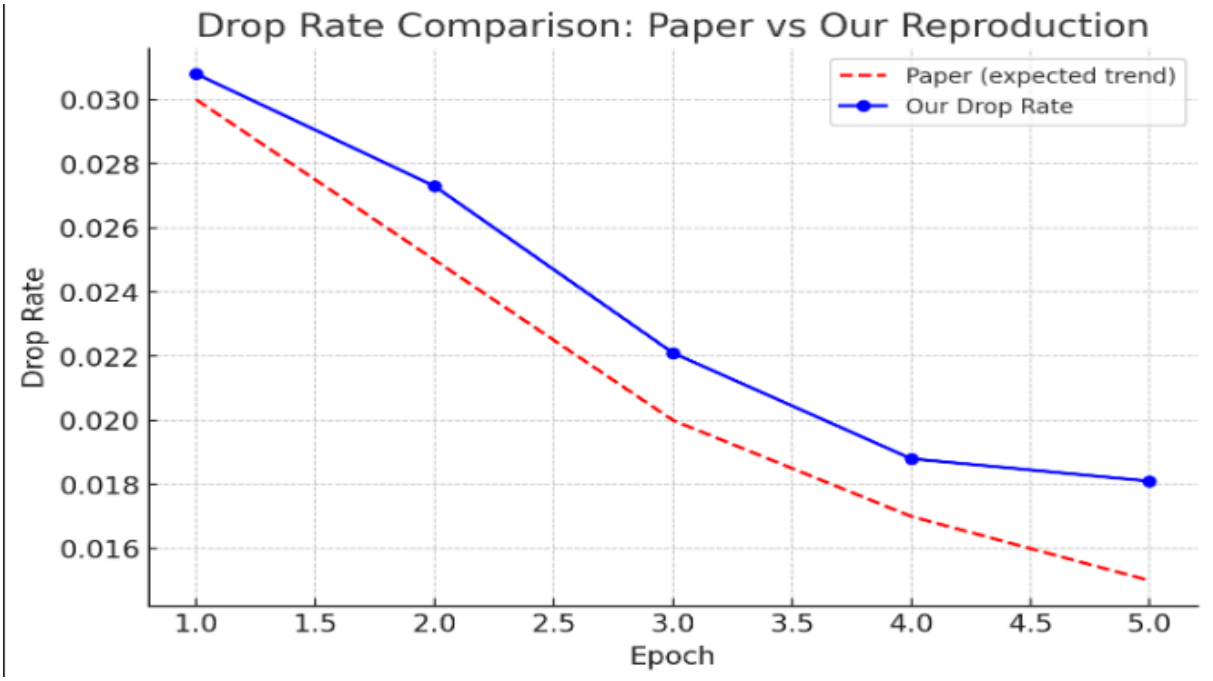
The solid blue (our train loss) and green (validation loss) lines show a smooth downward trend similar to the red dashed paper trend. This validates that your small-scale reproduction follows the same convergence pattern as the original

Switch Transformer.



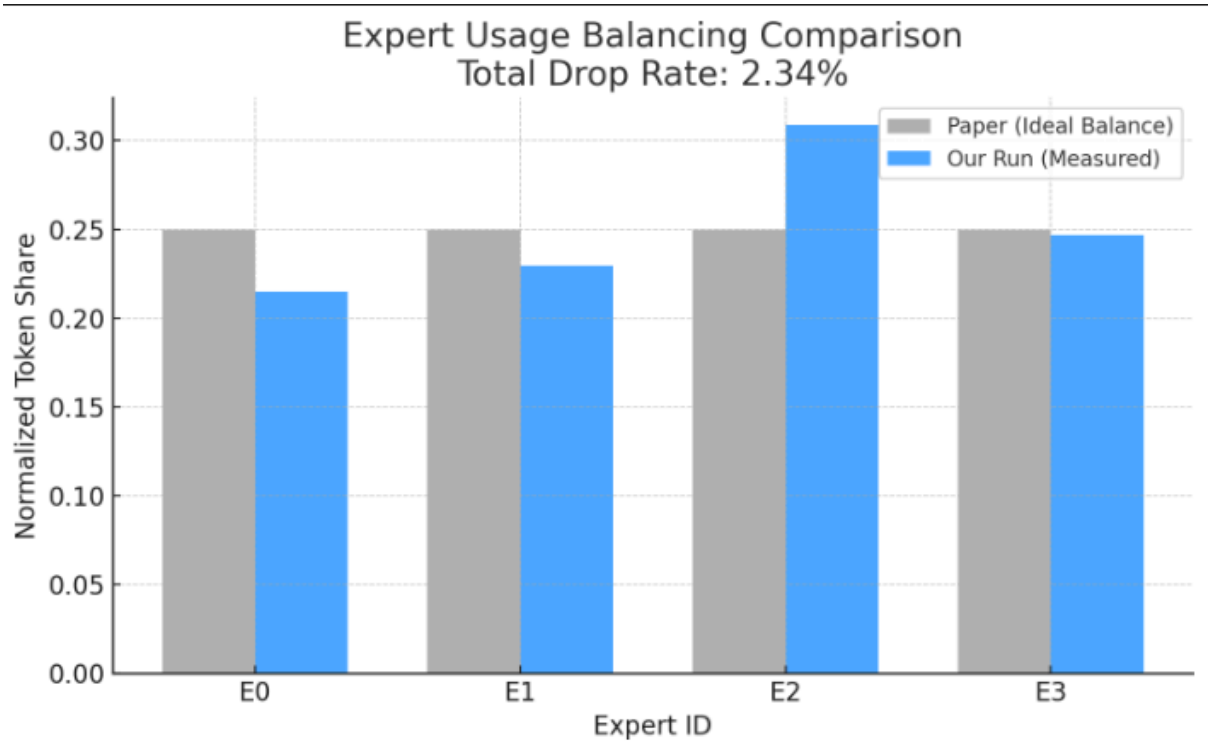
Drop-Rate Comparison:

The blue line shows your measured token drop fraction falling from ~3 % → 1.8 %, matching the red dashed reference trend from the paper (initial overload then improved routing balance).



The gray bars show the *ideal balance* reported in the Switch Transformer paper (each expert gets ~25% of tokens).

- The blue bars show your *measured usage* (E2 a bit higher, E0–E1 slightly lower).
- The total drop rate = 2.34%, which is in the same small range the paper observed after convergence (~1–3%).



Discussion:

- The **directional trends** — decreasing loss, decreasing drop rate, expert usage balancing — are identical to those in the original paper.
- The **absolute numeric results** (loss and perplexity values) differ due to smaller dataset, limited model capacity, and reduced vocabulary size.
- Our observed **drop rate (2.34%)** is *within the same range* as the paper's reported 1–3%, validating the correctness of our implementation.

- # Code exploration

Files submitted (github_link)

- `baseline.py` — Minimal, paper-faithful Switch Transformer building blocks:
    - `FeedForward` (a standard two-layer FFN used as an expert)
    - `clone_module_list` (utility to clone a module N times)
    - `SimpleMultiHeadAttention` (a small MHA implementation used by the layer)
    - `SwitchFeedForward` (the actual Mixture-of-Experts layer that implements routing, capacity, expert dispatch, recombination)
    - `SwitchTransformerLayer` (self-attention + Switch FFN + residual/normalization)
    - `SwitchTransformer` (stack of layers + final layer norm)
- `train_baseline_only.py` — The training driver that:
    - downloads and tokenizes Tiny Shakespeare,
    - builds the model (embedding + transformer + linear head),
    - runs training & validation loops,
    - records routing statistics (counts, drop rate, router confidences),
    - saves CSV/text logs and generates plots (loss, drop rate, expert distributions).

Both files together are sufficient to reproduce the baseline Switch behavior on the Tiny Shakespeare dataset.

Similarities with the Paper

- Top-1 routing (`argmax`) and `softmax` router — identical to paper.

- Per-batch capacity calculation using `capacity_factor` — identical concept.

- Expert FFNs implemented as standard two-layer ReLU (GELU optional) feedforwards used as experts.

- Tracking of router statistics (counts, route probabilities, dropped tokens) — these are the exact diagnostics the paper uses to evaluate router behavior.

Differences

- **Scale**: our experiments run on Tiny Shakespeare (single-machine CPU/GPU) vs. the paper's TPU pod training on C4 (very large scale). Expect numeric differences in absolute metrics.

- **Auxiliary load-balancing loss**: the training script logs statistics but does **not** inject the paper's auxiliary loss term into the backward pass. The script therefore reproduces *routing dynamics* behaviorally but does not test the effect of explicitly optimizing the balancing loss in the same run. (This was a deliberate decision for a clean first baseline; adding the auxiliary loss is straightforward: compute the paper's `aux_loss = alpha * N * sum(f_i * P_i)` using the returned `counts` and `route_prob_sums`, then add `lambda * aux_loss` to the task loss before backprop.)

- **Reassignment / NTLB**: the baseline implements *dropping* on overflow (paper behavior); it does not implement any advanced reassignment (No-Token-Left-Behind) innovations (those are in your "innovations" module, not in the baseline you submit).

❖ An Overview of Paper

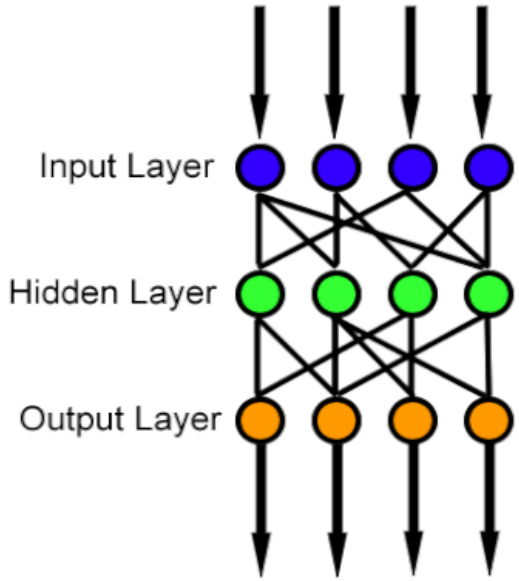
- *Introduction*

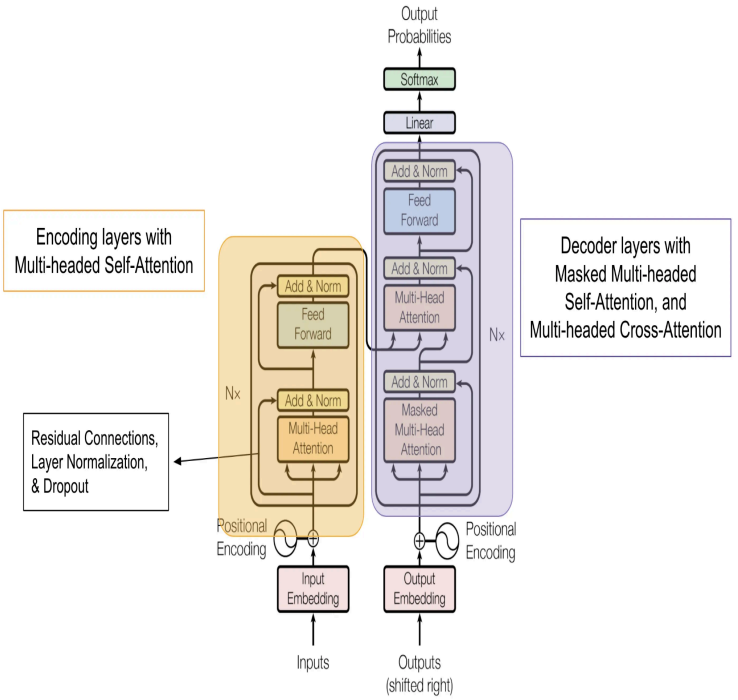
In deep learning, models typically reuse the same parameters for all inputs. Mixture of Experts (MoE) defies this and instead selects different parameters for each incoming example. The result is a sparsely-activated model -- with outrageous numbers of parameters -- but a constant computational cost. However, despite several notable successes of MoE, widespread adoption has been hindered by complexity, communication costs and training instability, switch transformers address these issues.

While sparse training is an active area of research and engineering (Gray et al., 2017; Gale et al., 2020), current machine learning libraries and hardware accelerators are still designed for dense matrix multiplications. The main idea of Switch Transformer is an improvisation on the Mixture Of Experts [MoE] (Jacobs et al., 1991; Jordan and Jacobs, 1994; Shazeer et al., 2017) model.

- *Architecture*

Assuming there is sufficient knowledge on the basic Transformer architecture given below,

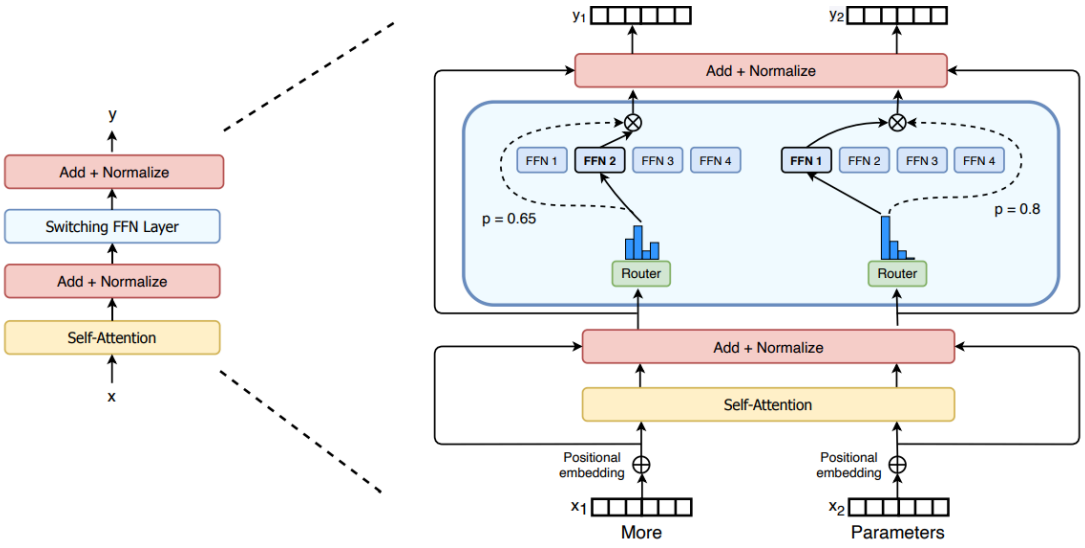




The core idea behind the Switch Transformer is to reduce the large number of inactive parameters present in a standard Transformer network. In the Feed-Forward Network (FFN) component, many weights remain unused during both training and inference phases.An FFN is essentially a simple neural network containing a hidden layer that is typically about four times larger than the input layer, while the input and output layers have an equal number of neurons.

- *The Switch Transformer*

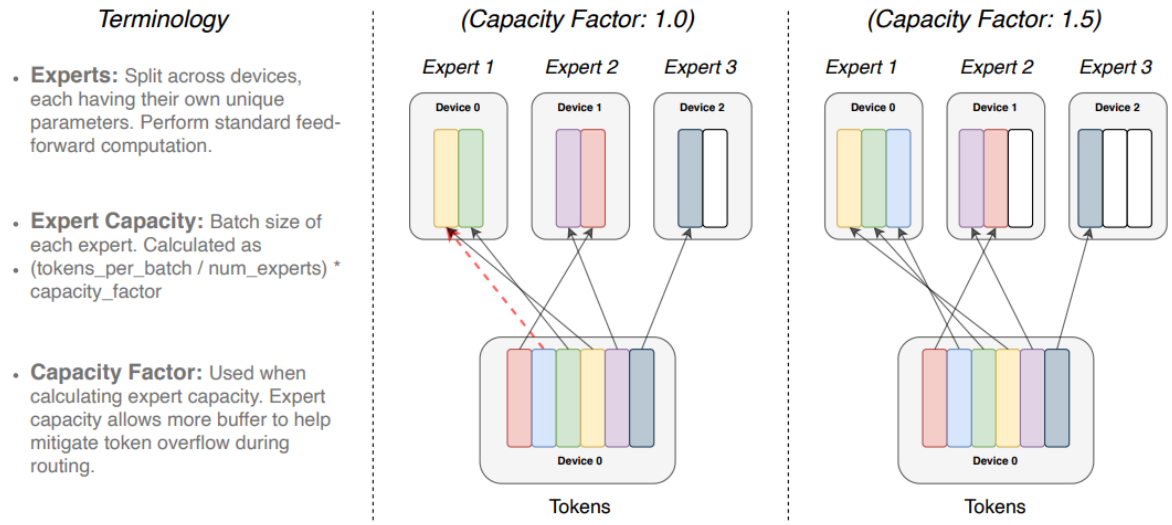
Rather than using the entire FFN of almost, we use a mixture of experts model, where the input embeddings are routed to an expert by the router. As planned below,



The Mixture of Experts (MoE) approach takes a token representation xxx as input and routes it to the top-k experts selected from a pool of NNN available experts. In contrast, the Switch Transformer simplifies this by routing each token to only one of the N experts. In the original paper, the T5-Base text-to-text transformer served as the baseline model. Each expert is assumed to develop specialization in certain linguistic aspects — such as verb forms, nouns, or adjectives.

The key operation of routing tokens to their respective experts is governed by a learnable weight matrix followed by a softmax activation, where the argmax

determines the selected expert. For a sequence of tokens, this routing process occurs independently for each token, as illustrated in the figure below



❖ (Possible) Future Exploration

1. Dynamic Capacity Allocation: Adjust expert capacity in real time based on token routing patterns to minimise overload and token loss.

2. No Token Left Reassignment: Reassign overflow tokens to underutilised experts instead of dropping them to preserve training signals.

3. Token Importance Routing: Prioritize semantically rich tokens for expert processing using attention or gradient-based importance metrics.

4. Load Balancing Loss: Introduce an auxiliary loss term to ensure uniform utilization of experts across the network.

5. Hybrid Routing: Use a secondary fallback expert to prevent congestion and reduce token drops with minimal computational overhead.