ANALYSE DÉTAILLÉE DE MIXTURE-OF-RECURSIONS (MOE)

Mokira
Ingénieur Machine Learning
(+229) 019 798 5109
dr.mokira@gmail.com

August 25, 2025

ABSTRACT

Pourquoi MoR est une petite révolution ? Imaginez un modèle de langage qui combine l'efficacité mémoire des architectures à partage de paramètres (comme ALBERT) avec l'intelligence computationnelle du calcul adaptatif (comme Mixture-of-Depths) : c'est exactement ce que propose Mixture-of-Recursions (MoR). Au lieu d'utiliser des couches distinctes, MoR réutilise un même bloc de calcul de manière récursive, tandis qu'un routeur léger décide dynamiquement pour chaque mot s'il doit « sortir » rapidement ou "réfléchir" plus longtemps en passant par des recursions supplémentaires. Résultat ? Une réduction simultanée de la taille du modèle (-50% de paramètres), du temps de calcul (meilleur débit d'inférence) et de la mémoire cache, sans sacrifier la performance — ouvrant la voie à des LLMs à la fois agiles, économiques et puissants. Une avancée architecturale majeure qui mérite d'être explorée dans les moindres détails !

1 Introduction

Aujourd'hui, les intelligences artificielles qui comprennent et génèrent du langage, comme celles qui animent les assistants virtuels, reposent sur des architectures dites « Transformers ». Si elles sont impressionnantes, leur formidable puissance a un coût : une gourmandise excessive en calcul et en mémoire. Pour fonctionner, ces modèles doivent en effet traiter chaque mot d'un texte avec la même intensité. Par exemple, pour comprendre un mot complexe comme « philosophique », un modèle de langue doit lui accorder autant de temps et de ressources qu'à un mot simple comme « le » ou « et ». Cette approche uniforme est coûteuse et inefficace.

Les LLMs sont puissants, mais gourmands en mémoire et en calcul. Et si on pouvait créer un modèle à la fois compact et intelligent, capable de concentrer ses efforts sur les mots qui en valent vraiment la peine ?

C'est la réponse à cette question qui a donné naissance à la **Mixture-of-Recursions (MoR)**. une nouvelle architecture qui permet à un modèle de langage d'allouer intelligemment son « effort de calcul » de manière adaptive, token par token. Imaginez une usine de traitement où les produits simples sont expédiés rapidement après une étape, tandis que les produits complexes passent par plusieurs stations de contrôle pour un travail

approfondi. MoR opère de la même façon : en réutilisant un même groupe de couches de neurones de manière recursive, et en utilisant un mécanisme de décision légère pour déterminer quels mots méritent plus de « réflexion ». Cette méthode unifie pour la première fois les gains en efficacité mémoire (moins de paramètres) et en efficacité computationnelle (moins de calculs superflus), sans compromettre les performances.

Dans ce didacticiel, nous commencerons par un rappel des concepts fondamentaux nécessaires à la compréhension. Ensuite, nous définirons précisément le problème qui se pose. Puis, nous détaillerons le fonctionnement de la solution proposée par MoR pour résoudre le problème, Nous illustrerons cette solution par des exemples concrets et une implémentation simplifiée en Python. Nous discuterons ensuite des apports majeurs et des limites de cette solution, et proposerons des pistes d'amélioration futures, et enfin conclure sur la portée de ce travail.

2 Connaissances de Base Nécessaires

Pour comprendre ce papier, il faut se familiariser avec quelques concepts clés des modèles de type Transformers.

2.1 Architecture Transformer de base (Vaswani et al., 2017)

Un transformer est un modèle de réseau de neurones qui utilise des **mécanismes d'attention** pour traiter des séquences de données (comme du texte). Il est composé de couches (layers) empilées les unes sur les autres. Chaque couche a une **auto-attention** et un réseau **feed-forward**.

Imaginez un Transformer comme une usine avec plusieurs étages (couches/layers). Chaque étage traite l'information et la passe à l'étage suivant. Dans un modèle traditionnel comme GPT :

Entrée : "Le chat mange" $\rightarrow$ Couche 1 $\rightarrow$ Couche 2 $\rightarrow \ldots \rightarrow$ Couche $N \rightarrow$ Sortie : "le poisson"

À chaque étage, chaque mot (token) regarde tous les autres mots pour mieux se comprendre soi-même (c'est l'auto-attention), puis une petite usine interne (le feed-forward) pour calculer sa représentation.

2.2 Cache KV (Key-Value Cache)

Pour générer du texte de manière autoregressive (un mot à la fois), on doit éviter de recalculer les représentations des mots passés à chaque nouvelle étape. Le cache KV stocke les "clés" et "valeurs" de tous les mots déjà générés, ce qui rend le processus beaucoup plus rapide.

De façon analogique, lorsque vous lisez un livre, vous n'oubliez pas les chapitres précédents. Votre cerveau garde en cache les informations importantes. Le cache KV fait la même chose pour le modèle.

2.3 Partage de Paramètres (Parameter Sharing)

Au lieu d'avoir des poids uniques pour chaque couche, on utilise le même jeu de poids pour plusieurs couches. Cela réduit énormément la taille du modèle.

Exemple : Au lieu d'avoir 24 stations de travail différentes dans notre usine, on n'en a que 8, mais on fait passer la phrase 3 fois par les mêmes 8 stations. L'usine est plus petite (moins de paramètres) mais le travail est tout aussi profond (3 passages = 24 traitements effectifs).

2.4 Calcul Adaptatif (Adaptive Computation)

L'idée que tous les mots ne méritent pas le même effort de calcul. Les mots de structure ("le", "la", "et") sont simples, tandis que les mots de fond ("quantique", "philosophie") sont complexes. Le calcul adaptatif permet au modèle de "réfléchir" plus longtemps aux mots difficiles.

3 Le Problème posé

Les modèles de langage (LLMs) comme GPT-4 sont très puissants mais ont deux énormes défauts :

- Ils ont des milliards de paramètres, ce qui les rend très coûteux à entraîner et à stocker.
- Le mécanisme d'attention standard a une complexité quadratique $O(n^2)$. Pour générer de longues séquences, le calcul devient extrêmement lent et gourmand en énergie.

Les solutions existantes s'attaquent souvent à un seul de ces problèmes à la fois.

- *ALBERT* réduit la taille du modèle mais applique le même nombre de calculs à tous les tokens (inefficient).
- *Mixture-of-Depths* réduit le calcul pour les tokens "faciles" mais n'utilise pas de partage de paramètres, donc la taille du modèle reste grande.

Le problème fondamental est donc : Comment créer un modèle qui est à la fois petit (grâce au partage de paramètres) et rapide à l'inférence (grâce au calcul adaptatif) sans sacrifier la performance ? C'est ce problème que MoR résout.

4 La Solution : Mixture-of-Recursions (MoR)

MoR est une architecture unifiée qui combine intelligemment le partage de paramètres et le calcul adaptatif.

4.1 Concept de Base

Au lieu d'empiler 24 couches différentes, MoR a un petit bloc de N couches partagées (par exemple, 8 couches). Ce bloc va être appliqué de manière répétée (récursive), jusqu'à N_r fois (par exemple, 3 fois). Le traitement effectif total est de $8 \times 3 = 24$ couches, mais on stocke que 8 couches de paramètres. Ce qui nous permet d'économiser de la mémoire stockage.

À la fin de chaque application du bloc récursif, un routeur léger décide pour chaque token s'il doit "sortir" (il a fini de réfléchir) ou s'il doit passer une nouvelle fois dans le bloc récursif (il a besoin de plus de réflexion). C'est le *Routage Adaptatif par Token* (voir figure 1). Cela permet d'économiser en calcul sur des mots simples à comprendre.

4.2 Token choice routing

Imaginez qu'avant même la course, on affecte un numéro de dossard à chaque coureur qui indique combien de tours il doit courir. Dès le début, un routeur regarde chaque token et lui assigne une "profondeur de

récursion". Prenons par semple la phrase suivante : `"Le chat philosophique mange la nourriture délicate."`, pour chaque mot on aura :

- `"Le"` —> 1 tour,
- `"Chat"` —> 2 tours,
- `"Philosophique"` —> 3 tours.

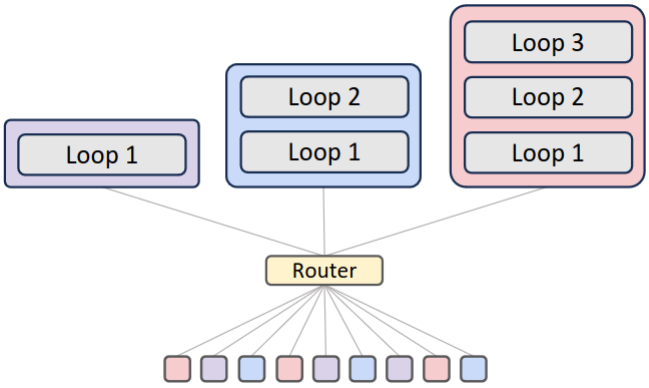

Figure 1: Routage de token [1]

Comme les tokens sortent à des étapes différentes donc ne traversent pas les même nombre d'étapes, on ne met en cache que les paires Clé-Valeur des tokens qui ne sont encore sortis des couches. On évite de stocker des résultats de calculs inutiles pour les tokens déjà sortis. Ce qui optimise la mémoire encore plus.

4.3 Headings: second level

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

$$\xi_{ij}(t) = P(x_t = i, x_{t+1} = j|y, v, w; \theta) = \frac{\alpha_i(t) a_{ij}^{w_t} \beta_j(t+1) b_j^{v_{t+1}}(y_{t+1})}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i(t) a_{ij}^{w_t} \beta_j(t+1) b_j^{v_{t+1}}(y_{t+1})} \tag{1}$$

4.3.1 Headings: third level

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Paragraph Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

5 Examples of citations, figures, tables, references

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui. [2, 3] and see [4].

The documentation for `natbib` may be found at

`http://mirrors.ctan.org/macros/latex/contrib/natbib/natnotes.pdf`

Of note is the command `\citet`, which produces citations appropriate for use in inline text. For example,

```
\citet{hasselmo} investigated\dots
```

produces

Hasselmo, et al. (1995) investigated. . .

`https://www.ctan.org/pkg/booktabs`

5.1 Figures

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi. See Figure 2. Here is how you add footnotes. [1] Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

[1] Sample of the first footnote.

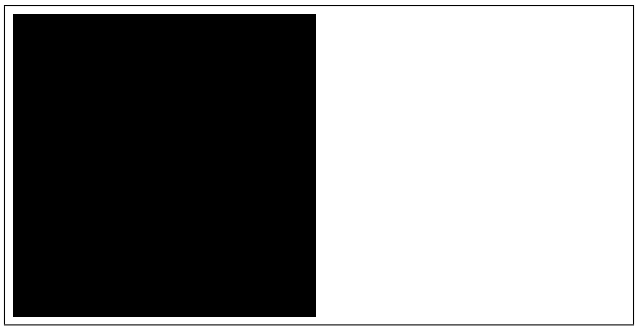

Figure 2: Sample figure caption.

	item_name	item_id	item_category_id
0	! ВО ВЛАСТИ НАВАЖДЕНИЯ (ПЛАСТ.) D	0	40
1	!ABBYY FineReader 12 Professional Edition Full...	1	76
2	***В ЛУЧАХ СЛАВЫ (UNV) D	2	40
3	***ГОЛУБАЯ ВОЛНА (Univ) D	3	40
4	***КОРОБКА (СТЕКЛО) D	4	40

5.2 Tables

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo. See awesome Table 1.

5.3 Lists

- Lorem ipsum dolor sit amet
- consectetur adipiscing elit.
- Aliquam dignissim blandit est, in dictum tortor gravida eget. In ac rutrum magna.

Table 1: Sample table title

Part		
Name	Description	Size (μm)
Dendrite	Input terminal	$\sim$100
Axon	Output terminal	$\sim$10
Soma	Cell body	up to 10^6

References

[1] Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation, 2025.

[2] George Kour and Raid Saabne. Real-time segmentation of on-line handwritten arabic script. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 417–422. IEEE, 2014.

[3] George Kour and Raid Saabne. Fast classification of handwritten on-line arabic characters. In *Soft Computing and Pattern Recognition (SoCPaR), 2014 6th International Conference of*, pages 312–318. IEEE, 2014.

[4] Guy Hadash, Einat Kermany, Boaz Carmeli, Ofer Lavi, George Kour, and Alon Jacovi. Estimate and replace: A novel approach to integrating deep neural networks with existing applications. *arXiv preprint arXiv:1804.09028*, 2018.