Deploying Programmatic Attention in Real Transformers

Recent work has shown that a large fraction of attention heads in Transformer language models can be replaced with programmatic proxies with modest cost to perplexity and little effect on downstream task performance. That result is framed as post-hoc interpretability where the programs are explanations. This proposal bridges the gap from explainability to deployment: what would it take for programmatic QK circuits to run inside real, usable Transformer systems, at minimal cost to both task performance and efficiency?

This is potentially one step towards more ambitious interpretability, where we train and deploy natively interpretable models — in this case, a neurosymbolic Transformer.

There are potentially separate roles these programs can play for inference vs training:

1. **For inference**: Programs are currently Python functions executed on CPU inside the forward pass, which is inefficient. there are two types of programmatic heads which I believe can be used for efficiency:
    a. Purely position-dependent programs (first-token / attention-sink, diagonal/local-window, strided, previous-token): these operate on the positions and can be directly integrated into existing efficient attention kernels.
    b. Lexical/content-dependent programs (attend to previous occurrence of the same token, to matching brackets/quotes, to sentence boundaries): these operate directly on the tokens, and can be expressed as a gather function over tokens

Because these heads are adaptive, compared to existing sliding-window or sparse attention heads that hold attention patterns fixed for context, they may even offer additional performance-per-unit-efficiency gains.

2. **For training**: Could we initialize models with (a subset of) these programmatic heads and then train against them, and does this offer faster training convergence? Recent work shows that models converge quicker if bootstrapped with task-relevant attention patterns (and a large component of grokking is the sudden emergence of these attention patterns). This suggests programmatic attention is useful not only as a description of trained models but as an initialization and constraint for training them.
    a. The most salient question is figuring out how programmatic heads should be efficiently integrated into training (see 1, where computing these programmatic proxies must be computed inefficiently on CPU)
    b. As a stretch, could we design an iterative training approach where we train the model, select and fix a subset of attention patterns, retrain, re-fix, etc., until we obtain a maximally performant neuro-symbolic transformer, where all QK patterns are programmatic proxies
    c. Another stretch could be predicting these patterns from the task itself: Train an LM-driven synthesis agent, given only the task specification and example sequences, can propose the appropriate attention programs, closing the loop from task → program → initialization.