Adaptive Computation Modules: Granular Conditional Computation for Efficient Inference

Bartosz Wójcik[1, 2], **Alessio Devoto**[3], **Karol Pustelnik**[4], **Pasquale Minervini**[5, 6], **Simone Scardapane**[3]
[1]IDEAS NCBR
[2]Jagiellonian University
[3]Sapienza University of Rome
[4]University of Warsaw
[5]University of Edinburgh
[6]Miniml.AI

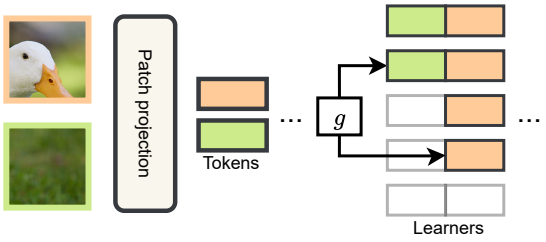
Abstract

While transformer models have been highly successful, they are computationally inefficient. We observe that for each layer, the full width of the layer may be needed only for a small subset of tokens inside a batch and that the "effective" width needed to process a token can vary from layer to layer. Motivated by this observation, we introduce the *Adaptive Computation Module* (ACM), a generic module that dynamically adapts its computational load to match the estimated difficulty of the input on a per-token basis. An ACM consists of a sequence of *learners* that progressively refine the output of their preceding counterparts. An additional gating mechanism determines the optimal number of learners to execute for each token. We also propose a distillation technique to replace any pre-trained model with an "ACMized" variant. Our evaluation of transformer models in computer vision and speech recognition demonstrates that substituting layers with ACMs significantly reduces inference costs without degrading the downstream accuracy for a wide interval of user-defined budgets.

Introduction

Driven by their constantly improving capabilities, state-of-the-art neural networks have been experiencing a continued growth in size in the last decade (Pugliese, Regondi, and Marini 2021). This progress was made possible by algorithmic and model design improvements (in particular with the introduction of transformer models) and the increasing computational power of modern GPUs. Scaling up the model architecture frequently results in improved performance (Devlin et al. 2018; Zagoruyko and Komodakis 2016), causing the size and computational costs of state-of-the-art models to keep increasing steadily. These escalating costs limit applications in latency-sensitive and energy-constrained scenarios and contribute to higher carbon emissions, exacerbating environmental concerns (Schwartz et al. 2020; Patterson et al. 2022).

Several approaches from the literature aim to mitigate this problem. For example, quantization reduces inference time by quantizing the weights and activations into low-precision floating-point (Courbariaux, Bengio, and David 2014) or integer values (Wu et al. 2020; Dettmers et al. 2022). Knowledge distillation methods (Hinton, Vinyals, and Dean 2015; Aguilar et al. 2020) can transfer knowledge from an ensemble or single larger model into a smaller network. Finally, sparsification methods (LeCun, Denker, and Solla 1989; Han et al. 2015; Hoefler et al. 2021) yield either sparse weight or sparse activation tensors.



(a) High-level overview of the Adaptive Computation Module

(b) Example images with computation spatial load maps

Figure 1: ACMs adapt their computational load for each input on a per-token basis by selecting the number of *learners* to execute via a trainable gate. In the example on the top, a background token (green) is allocated fewer learners than a content-rich token (orange). This results in a spatially varying computational load, as shown on the bottom.

While they incur a slight decrease in downstream model performance, such methods show that some neural modules can often be replaced with computationally cheaper alternatives. Based on this observation, we argue that in transformer models (Vaswani et al. 2017), the representational capacity of a whole layer is not needed for every input token. In other words, we hypothesize that the same transformation could be achieved in a more computationally efficient manner. To explore this hypothesis, we introduce *Adaptive Computation Modules* (ACMs), a neural network module that adapts its computational burden to match the difficulty of the current input token. An ACM consists of a sequence

of *learners* and a single gating network. The task of each learner is to improve upon the combined output of previous learners, while the gating network determines the optimal number of learners to execute for each input token. Since all token-level decisions are independent, the resulting model is a highly granular application of the conditional computation paradigm (Bengio 2013; Bengio et al. 2015), and thus allows for spatial adaptability in transformer models (Figurnov et al. 2017; Han et al. 2021). Figure 1 provides an outline of the proposed model.

To enable the use of a vast range of publicly available models, we propose a straightforward conversion procedure in which we: (1) substitute the selected blocks of the model with ACMs, (2) initialize the learners by distilling knowledge from the substituted blocks, (3) pre-train the gating networks using artificially generated labels, and (4) train the entire model in an end-to-end manner.

Our approach can significantly decrease the model's computational footprint without sacrificing performance. We evaluate our method on the ImageNet-1k (Russakovsky et al. 2015) dataset with Vision Transformer (ViT) models and on speech recognition with pre-trained Wav2Vec networks (Baevski et al. 2020), and show that in both cases we achieve a better performance-efficiency trade-off than other conditional computation methods. We make the following contributions:

- We show that individual layers in transformer models can be computationally inefficient and that their entire expressiveness is often needed only for a small subset of input tokens.
- We introduce ACM, an easy-to-train, modular, and general-purpose module that offers a granular approach for reducing the computational cost of transformers.
- We propose a smart initialization strategy for the parameters of ACMs that relies on module-wise knowledge distillation from pre-trained models.
- We provide an efficient GPU implementation to demonstrate that ACMs effectively speed up inference.

Related Work

We provide a brief overview of related works, focusing on our experimental comparison. A longer overview with a broader outlook can be found in the supplementary material.

Conditional Computation

Conditional computation (CC) refers to the ability of a model to adapt its computational graph to its input. While CC can be used for problems such as continual learning (Lin, Fu, and Bengio 2019), most CC methods adjust their execution on a per-input basis to significantly reduce their average computational cost while maintaining performance (Scardapane et al. 2024).

In the following, we focus on CC algorithms that can be applied to any pre-trained transformer model with minimal modifications; architecture-specific CC algorithms (e.g., dynamic channel selection for CNNs (Chen et al. 2019; Li et al. 2021)) are discussed in the supplementary material. We broadly categorize the methods into three groups: early-exits (EEs) (Teerapittayanon, McDanel, and Kung 2016; Bolukbasi et al. 2017), mixture-of-experts (MoEs) (Yuksel, Wilson, and Gader 2012; Shazeer et al. 2017; Fedus, Dean, and Zoph 2022), and token dropping (TD) (Rao et al. 2021; Yin et al. 2022; Meng et al. 2022; Haurum et al. 2023). Finally, Cai et al. (2024) is a concurrent conditional computation work that is remarkably similar to the proposed ACMs. In particular, it also trains a router for each layer and converts a pre-trained dense model with a three-step method.

Early-exits

In EE models, inputs are allowed to "exit" the architecture at intermediate layers via additional classifier heads that are trained together with the backbone network (Teerapittayanon, McDanel, and Kung 2016), or as a separate phase in a layerwise fashion (Han et al. 2021), and the predictions of multiple heads can be merged with a number of different techniques (Teerapittayanon, McDanel, and Kung 2016; Wołczyk et al. 2021; Scardapane et al. 2020). In EE networks, the execution is stopped for the entire input at once, as opposed to individual tokens as is done in the proposed ACMs. In addition, designing and placing exit heads is itself a non-trivial problem (Teerapittayanon, McDanel, and Kung 2016), and very little work has been done outside standard computer vision and natural language processing classification tasks. In contrast, ACMs are designed to replace individual blocks in a transformer model in a plug-and-play fashion.

Token Dropping

TD strategies either prematurely stop execution (Rao et al. 2021; Yin et al. 2022; Haurum et al. 2023) or skip computation of selected layers (Meng et al. 2022) for *individual tokens* that are deemed less relevant or redundant. We can interpret them as dynamically allocating a variable depth to each token. Our proposed ACM can be seen as an orthogonal *width* variant of token dropping, in which each token is allocated a variable *width* for each layer in the network.

Mixture-of-Experts

In MoEs, selected modules of a model are replaced by an independent set of blocks called experts, which are selectively activated for each token by an additional routing network (Puigcerver et al. 2023). MoEs have been successfully developed for replacing MLP layers in transformer models (Shazeer et al. 2017; Riquelme et al. 2021), attention layers (Zhang et al. 2022), entire blocks (Tan et al. 2023), and adapters (Zadouri et al. 2023). Although MoEs are typically trained from scratch or fine-tuned from existing models (Zadouri et al. 2023), a small number of works have investigated *moefication* procedures (Zhang et al. 2021; Qiu, Huang, and Fu 2023). Importantly, in MoEs, each token is allocated a fixed amount of compute depending on the routing strategy (e.g., top-k routing (Riquelme et al. 2021)). ACM can be seen as a modification of MoEs in which the experts are ordered, and thus, the complexity of the routing problem is drastically reduced. The gating network decides *how many* learners instead of *which* experts to execute,

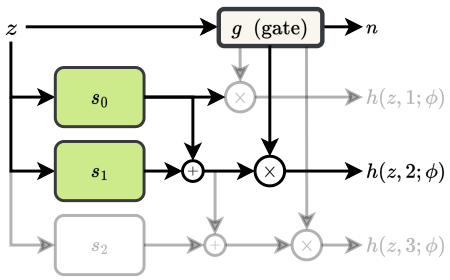


Figure 2: Architecture of an ACM block: the output is the sum of k *learners*, where k is determined on a per-token basis by a small gating network g. The learners are executed in parallel. In the example, only the first two learners are executed, and the computation of the third (greyed out) is skipped.

therefore allowing for a variable amount of computation on a per-token basis. A concurrent work of Jain et al. (2024) defines nested experts, which results in a dynamic model similar to ACMs.

Method

An ACM is a conditional computation block that adapts its execution cost for each processed token via a trainable gating layer. In this paper, instead of training an ACM-based model from scratch, we focus on converting *any* pre-trained transformer network into an equivalent "ACMized" version, i.e. one having similar accuracy while achieving a pre-defined computational budget on average. To this end, we propose an effective weight pre-initialization scheme. First, we substitute a subset of layers of the base model (e.g., MLPs or MHA projections) with an ACM of similar size. The ACMs are then trained independently but *in parallel*, first with a per-layer reconstruction loss to initialize the learners and then using a cross-entropy loss to initialize the gates. The actual training consists of fine-tuning the model in an end-to-end manner to allow it to adapt its weight to the dynamic inference setting. In the supplementary material, we demonstrate that this setup significantly speeds up the training of the ACMized networks in all cases.

Adaptive Computation Module

Our ACM module aims to allow the model to work well with any required computational budget while still ensuring efficient execution on GPUs, a property most of the dynamic width methods lack (Li et al. 2021). Given an input token z, consider a set of N homogeneous modules, $s_n(z; \phi_{(n)})$, $n \in \{1, ..., N\}$, each module having weights $\phi_{(n)}$. We refer to these modules as *learners*. In an ACM, each learner progressively refines the prediction of the previous ones such that the k-th output of the ACM block, $k \in \{1, ..., N\}$, is given by:

$$h(z, k; \phi) = \sum_{n=1}^{k} s_n(z; \phi_{(n)}) \tag{1}$$

All intermediate outputs $h(z, 1; \phi), \ldots, h(z, N; \phi)$ are valid choices for the output of the ACM: a larger value for k yields a more accurate result at the cost of a higher computational burden, while $k = 1$ means that only a single learner is executed. Note that once k is known for a token, the learners can be executed in parallel.

For any token z, k should be chosen as the smallest possible value that guarantees good network performance. To this end, at the beginning of the ACM block, we add a small trainable gating network g with weights ω to select the number of learners k to be executed. In practice, the gating network returns N real-valued outputs, which are then discretized into a one-hot gating choice vector with the Gumbel-Softmax trick (Jang, Gu, and Poole 2016) to retain differentiability:

$$\nu_n = \frac{\exp((\log(g(z;\omega))_n + \gamma_n)/T)}{\sum_n \exp((\log(g(z;\omega))_n + \gamma_n)/T)}. \tag{2}$$

In Equation (2), $\gamma_1, ..., \gamma_N$ are i.i.d samples from $\texttt{Gumbel}(0, 1)$, and T is softmax temperature. In the forward pass, we discretize ν into a one-hot vector $\hat{\nu}$, but the continuous values are used directly for gradient computation in the backward pass. The complete ACM architecture is outlined in Figure 2.

Design of the ACM blocks Any network architecture with the required input and output dimensionality can play the role of a learner. For simplicity, in this paper, we always use two dense layers with a GELU activation function in between. Since we replace selected modules of a pre-trained model with ACMs, we always pick N and the size of a single learner such that the entire ACM module has approximately the same computational cost and the number of parameters as the substituted block. This is straightforward to achieve as the cost of a learner scales linearly with its hidden dimensionality. However, the output dimension has to be the same as in the replaced module. This results with output layer biases taking a larger share of learner parameters as we increase N. To avoid this effect, we simply eliminate them from our architecture.

For the gating network, we also use a two-layer network and determine its hidden dimensionality such that the total computational cost of gating is around 1% of the overall model cost. When feasible – such as the module being placed under a residual connection – we set the minimum number of executable learners to 0 so that even the computation of the first learner can be skipped.

ACM weight initialization

The costs of training large models are constantly increasing (Strubell, Ganesh, and McCallum 2019). On the other hand, there is a large number of publicly available pre-trained models, so the capability of adapting them is a desired property for new methods. Since ACMs are designed to replace individual modules, we initially train them by distilling the knowledge from the corresponding trained modules that are being substituted. Specifically, we propose the following scheme. A trained static model $f(\cdot)$ is cloned, and selected modules (e.g., every MLP module) are replaced with

ACMs in that cloned model $f_{\text{ACMized}}(\cdot)$. Each learner from every ACM is initialized randomly. For each sample x_i from a mini-batch B forwarded through the original model, we save every input token $z^l_{i,j}$ and output token $o^l_{i,j}$ of each l-th module that was replaced in the copied model. Then, every ACM is trained independently and in parallel by minimizing the mean squared error (MSE) applied for every possible choice of k:

$$\mathcal{L}(\phi) = \frac{1}{|B|SLN} \sum_{i,j,l,n} \left\| h(z^l_{i,j}, n; \phi^l) - o^l_i \right\|^2 \quad (3)$$

where S is token sequence length, and L is the number of substituted modules. Note that the gating network is not needed in this phase. The effectiveness of this training approach can be tested by setting a fixed k for every ACM in the model and evaluating the model on the validation set.

With learners being able to reliably imitate the replaced modules, we subsequently freeze them and train the gating networks in a similar, layer-wise approach. We frame the problem as a classification task and generate artificial labels with the following heuristic. First, we consider the outputs of all learners and compute the distance to the original output:

$$d^l_{i,j}(n) = \| h(z^l_{i,j}, n; \phi) - o^l_{i,j} \|_2 \quad (4)$$

The target label is then set as:

$$t^l_{i,j} = \min \left\{ n \in \{2, ..., N\} \;\middle|\; \frac{d^l_{i,j}(n)}{d^l_{i,j}(n-1)} \geq \tau \right\}, \quad (5)$$

where τ is a threshold hyperparameter. In other words, we select the smallest number of learners such that the relative improvement from adding one more learner is lower than the threshold τ. With these labels, the gating networks are trained using standard cross-entropy loss:

$$\mathcal{L}(\omega) = \frac{1}{|B|SLN} \sum_{i,j,l,n} -t^l_{i,j,n} \log(\nu^l_{i,j,n}). \quad (6)$$

End-to-end training

In the third phase, we finetune the entire model end-to-end to allow it to adapt its weights to the dynamic inference setting. To make the model more predictable in terms of its computational cost, we add an auxiliary loss term that penalizes for any deviation from the given target budget β_{target} on average:

$$\mathcal{L}_{\text{b}}(\theta) = \left\| \frac{1}{|B|} \sum_i \frac{\sum_j \sum_l k^l_{i,j} p^l}{\sum_j \sum_l N p^l} - \beta_{\text{target}} \right\|_1, \quad (7)$$

where p^l is the computational cost of a single learner from layer l and $\beta_{\text{target}} \in (0, 1)$ is the targeted computational budget. This term still allows for the allocation of different amounts of computational resources to samples of varying complexity and only requires to be close to β_{target} *on average*. It also affects the computational cost of future training steps, potentially accelerating the training process.

While $\mathcal{L}_{\text{b}}$ does not prevent diversity of gating choices, it does not encourage it. In practice, we find that the gating networks collapse to a state where the same number of learners is chosen for every input token, effectively turning our dynamic model into a static one. To prevent this behavior and encourage diversity, we add two additional loss terms. The first auxiliary loss term maximizes the average normalized entropy of gating choices taken for tokens in a single image:

$$\mathcal{L}_{\text{e}}(\theta) = \frac{1}{|B|L} \sum_{i,l} \frac{\sum_n a_n \log(a_n)}{\log(N)}, \quad (8)$$

where:

$$a^l_{i,n} = \frac{\sum_j \hat{\nu}^l_{i,j,n}}{\sum_n \sum_j \hat{\nu}^l_{i,j,n}} \quad (9)$$

represents a distribution between N choices in batch B for the l-th ACM aggregated for an entire sequence of tokens from sample i. The intuition behind this loss is that not every input region is equally important; hence, a non-uniform distribution of computation is required.

Finally, entire images may exhibit different difficulty levels, and enforcing diversity of gating choices for a single image at a time may not be enough. We address this with the second auxiliary loss, which is defined as:

$$\mathcal{L}_{\text{d}}(\theta) = \frac{1}{|B|^2} \sum_i \sum_m \| b_i - b_m \|_1, \quad (10)$$

where:

$$b_i = \frac{\sum_j \sum_l k^l_{i,j}}{\sum_j \sum_l N} \quad (11)$$

denotes the fraction of learners executed on sample i. It encourages the model to distribute its computational budget between easy and hard examples. The final loss that is minimized for classification tasks is given by:

$$\begin{aligned} \mathcal{L}(\theta) = & \frac{1}{|B|} \sum_i \sum_c -y_{i,c} \log(\hat{y}_{i,c}) + \alpha_{\text{b}} \mathcal{L}_{\text{b}}(\theta) \\ & + \alpha_{\text{e}} \mathcal{L}_{\text{e}}(\theta) + \alpha_{\text{d}} \mathcal{L}_{\text{d}}(\theta), \end{aligned} \quad (12)$$

where $\{(x_i, y_i)\}_{i=1}^{|B|}$ are samples from the current mini-batch B, and α_{b}, α_{e}, α_{d} are hyperparameters for weighting the different terms. In practice, we always use $\alpha_{\text{b}} = 0.1$, $\alpha_{\text{e}} = 0.05$, and $\alpha_{\text{d}} = 0.05$. In the analysis section, we present an ablation study that demonstrates the necessity of applying the proposed auxiliary losses and shows the positive impact of their interplay. Note that the auxiliary losses are task-agnostic, allowing ACMs to be used for any other task than classification.

Experiments

Due to the widespread availability of pre-trained weights, we evaluate our method on popular image classification and speech recognition tasks. The aim of the evaluation is to compare the performance-efficiency trade-offs of different methods. To measure the computational efficiency, we track the number of FLOPs used by the model for each evaluated sample and report the averaged result. The source code for our experiments is available at https://github.com/bartwojcik/adaptive_computation_modules.

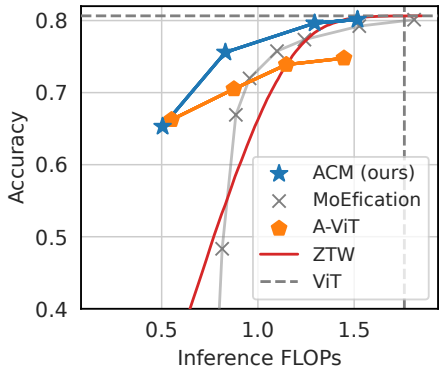


Figure 3: Performance-efficiency trade-offs of different conditional computation methods as measured on the ImageNet-1k dataset. ACM-based ViT-B achieves the Pareto frontier for a wide range of computational budgets.

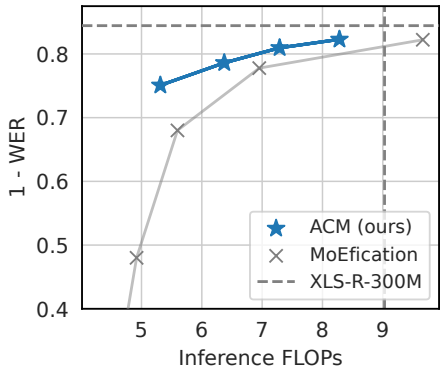


Figure 4: Performance-efficiency trade-offs of different conditional computation methods as measured on the CommonVoice-es dataset. The model's performance is reported in terms of Word Error Rate (WER). ACMs achieve lower WER for every evaluated computational budget.

Computer Vision

We select a ViT-B (Dosovitskiy et al. 2020) model pre-trained on ImageNet-1k (Russakovsky et al. 2015) from the `torchvison` library[1] as the base model for all methods. We compare ACMized models with three conditional computation techniques, each one coming from a different group: A-ViT (Yin et al. 2022) (Token Dropping), MoEfication (Zhang et al. 2021) (Mixture-of-Experts), and Zero Time Waste (Wołczyk et al. 2021) (Early Exiting). For the sake of a fair comparison, we assume the same training data budget of 100 epochs for every method.

Since MoEfication requires models using ReLU activation functions, we first replace GELU activations with ReLUs and finetune the model for 80 epochs, as described by Zhang et al. (2021). The remaining 20 epochs are used for training the routers. For Zero Time Waste, we train the early-exit heads for 95 epochs and then train the ensembles for 5 epochs. For the ACMized ViT, we replace every MLP module and every linear projection in each MHA with an ACM module (the self-attention mechanism itself is not affected). Module-wise representation distillation is performed for 2 epochs, followed by 1 epoch for pre-training of the gating networks and 97 epochs of end-to-end finetuning of the entire model. We set the number of learners N to 4 in every ACM. While MoEfication and Zero Time Waste can be evaluated with different numbers of selected experts and early exit confidence thresholds, A-ViT and ACMs need to be trained multiple times with different values of hyperparameters that approximately determine the final average computational budget. We emphasize that our A-ViT implementation includes fixes for two issues raised by GitHub users[2][3], which may affect the final performance of A-ViT. The authors of A-ViT have not yet addressed them at the time of writing this article.

[1]https://pytorch.org/vision/stable/models.html
[2]https://github.com/NVlabs/A-ViT/issues/4
[3]https://github.com/NVlabs/A-ViT/issues/13

We present the results in Figure 3. As projections in MHA are responsible for around 30% of the model cost, MoEfication seems to be hindered by the fact that it reduces only the cost of the MLP layers. A-ViT shows a gap in performance in relation to the pre-trained model, while Zero Time Waste is competitive only for higher computational budgets. ACMized ViT displays favorable performance for a wide range of computational budgets, and its advantage is especially significant below 12.5 GFLOPs.

Speech-to-Text

For speech recognition tasks, we use the XLS-R-300M (Babu et al. 2021), a pre-trained Wav2Vec2 model (Baevski et al. 2020), fine-tuned on selected languages from the CommonVoice dataset (Ardila et al. 2020). Speech-to-text models introduce additional complexities for dynamic computation methods, as each token is decoded individually. Since A-ViT and Zero Time Waste were not designed for this setting, we restrict our baseline methods to MoEfication, the only task-agnostic method among those three.

We assume an equal training data budget of 10 epochs for all approaches. In the case of MoEfication, we substitute GELUs with ReLUs and train for eight epochs, followed by two epochs of router training. For ACMs, we replace every MLP block within the model with an ACM module with $N = 4$. We subsequently perform five epochs of module-wise distillation, one of training the gating networks, and four epochs of end-to-end finetuning. We present results in Figure 4. We see that even if only the MLP blocks are ACMized, our method still obtains a better performance-efficiency trade-off than MoEfication.

Analysis

Dynamic models allocate variable amounts of compute for different samples or input regions. In this section, we provide motivation for our method, examine the distribution of computational load, and show that the introduced auxiliary

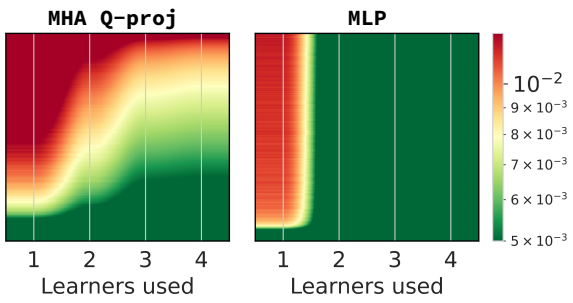


Figure 5: Color-coded errors of a 4-learner ACM plotted after performing module-wise representation distillation for modules from eight block of a ViT-B pre-trained model. Tokens are sorted along the Y-axis of the plot by their average error. For most input tokens, the same transformation can be performed by a considerably smaller module consisting of only two or three learners, thus justifying the use of ACMs.

losses are needed. In the supplementary material, we show the impact of the proposed pre-training stages and investigate the effect of changing the number of learners N.

Computational Inefficiency of Static Modules

To justify the use of ACMs, we make the following experiment. First, we perform module-wise distillation from a selected static module of an ImageNet-1k pre-trained ViT-B into 4 learners, just as we describe in the Method section. After convergence, we randomly select 5000 sample images from the validation dataset and forward them through the original model to save every input token $z_{i,j}$ and output token $o_{i,j}$. For every possible number of learners used $n \in \{1, ..., N\}$, we are interested in how well the learners imitate the output of the original module. For this, as in training, we use MSE: $\left\|h(z^l_{i,j}, n; \phi^l) - o^l_i\right\|^2$. The resulting tensor of MSE values has size $(5000, 197, 4)$, where 197 is the sequence length specific to ViT-B with patch size set to 16. Since ACMs process each token independently, we flatten the first two dimensions and then sort the tokens by the average error for readability. Figure 5 shows the resulting color-coded error visualizations for selected modules. We emphasize that the four learners have approximately the same cost and number of parameters as the original static module being substituted.

The results show that 1) only a small subset of tokens require the full processing power of the entire module, and for a majority of tokens, and 2) tokens usually exhibit varying levels of difficulty, thus warranting the use of conditional computation on a per-token basis.

Qualitative Analysis

A dynamical model should take advantage of the fact that images exhibit different difficulty levels for classification. By extracting the gating choices being done by every ACM for each patch, we can plot a heatmap that indicates which regions the model was focused the most on. This map should correlate with the meaningful regions of the image. Figure 6 shows that the model indeed learns to allocate its computational budget to those regions. We show that this effect is not exclusive to vision by performing a similar analysis for audio in Figure 7. We provide additional examples in the supplementary material.

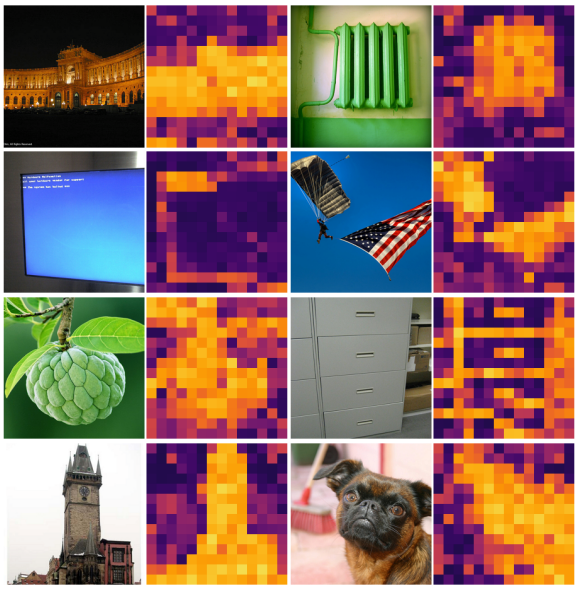


Figure 6: Computational load heatmaps for the model trained with $\beta_{\text{target}} = 0.6$. The model allocates its computational budget to semantically meaningful patches.

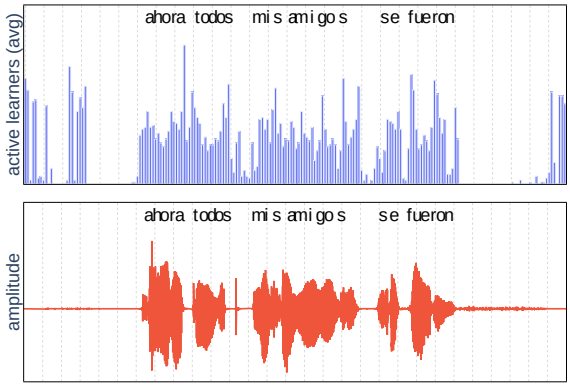
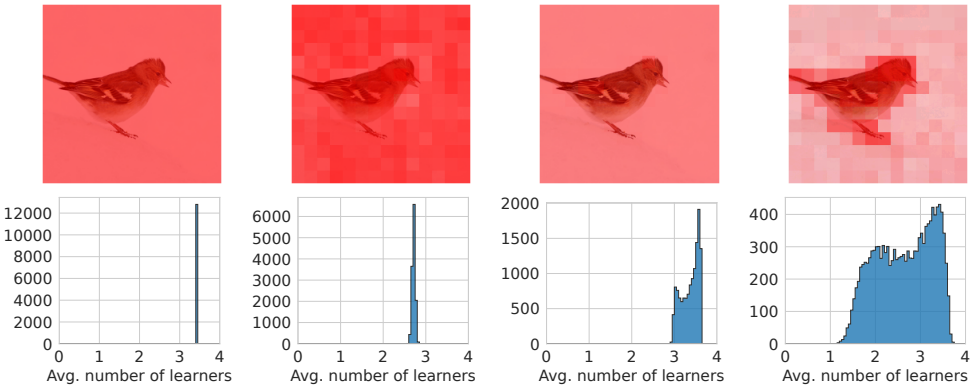


Figure 7: For each input audio token (red waveforms), we show the average number of learners that were activated in the ACMized model for $\beta_{\text{target}} = 0.25$ (blue bars). We can see that this model also learned to allocate its computational budget to important regions of the input.

Ablation Study

Being able to dynamically allocate computation when solving a task is the core feature of ACMized models. The auxiliary losses are necessary for the dynamism to emerge. We empirically demonstrate this in Figure 8 by training multiple runs differing only in the loss terms applied during the



(a) Only $\mathcal{L}_b$ (b) $\mathcal{L}_b$ and $\mathcal{L}_e$ (c) $\mathcal{L}_b$ and $\mathcal{L}_d$ (d) $\mathcal{L}_b$, $\mathcal{L}_e$ and $\mathcal{L}_d$

Figure 8: Effects of training with different combinations of enabled auxiliary losses. The computational load heatmap is rendered over the original image (top row). The histograms (bottom row) show the distribution of the total computational cost spent by the model on images from the validation set. Only combining all the proposed terms provides the desired diversity of the allocated computational budget.

end-to-end training phase. For each run, we visually analyze their intra-sample computational load distribution and generate a histogram of the total computation spent on every sample.

As expected, ACMs may converge to a solution in which always the same learners are used when only $\mathcal{L}_b$ is applied. The inclusion of $\mathcal{L}_e$ helps in diversifying between different regions of the input, but overall the computation spent on each image is mostly the same. Applying $\mathcal{L}_d$ instead of $\mathcal{L}_e$ diversifies the distribution of compute spent on every image, but the gating choices for different regions of the input are not diverse on its own. Only by enabling all the proposed losses can we achieve the desired effect of the model focusing on semantically meaningful patches and having a high variability of computational budget allocation.

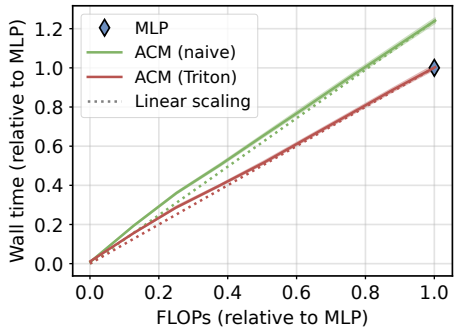


Figure 9: Latency of a 128-sample batch processed by a single ACM layer on an A100 GPU. The gating network is included in the measurements, but we replace the gating choices with samples from a random distribution to achieve the desired average number of executed learners. ACMs have negligible overhead, and latency scales linearly with the average number of executed learners.

Hardware speedup

Due to their design, ACMs are inherently well-suited for parallel execution on GPUs. Specifically: (1) once gating decisions are determined, the execution of learners can occur in parallel; (2) tokens can be reordered without any additional copying when they are loaded from or stored to the main GPU memory by the kernel; (3) when tokens are sorted by the number of selected learners, the computation for each group is standard matrix multiplication for which GPUs are highly optimized for; (4) the hidden dimensionality of each learner is deliberately large to maximize GPU utilization. We implement the ACM forward pass with GPU kernels written in Triton (Tillet, Kung, and Cox 2019) and employ several optimizations including configuration autotuning and kernel fusion.

In Figure 9 we evaluate our implementation by measuring the wall-clock time of execution of a single ACM module from the ViT-B-based model. The overhead in respect to a static MLP is negligible. Moreover, the wall clock time decreases linearly with the number of executed learners.

Conclusion

In this work, we have demonstrated that large static modules lead to ineffective allocation of computational budget. The introduced ACM, a generic module that facilitates granular conditional computation, is designed for computational effectiveness, and models based on it achieve state-of-the-art results among the tested dynamic inference methods. Our training procedure distills a static network into an adaptive one, forcing it to allocate its computational budget to semantically meaningful input regions. Future work might explore the relationship between ACMs and pruning methods. Since our training procedure replaces individual modules with ACMs, one could also consider using quantized learners for further efficiency gains.

Acknowledgements

Bartosz Wójcik is supported by National Centre of Science (NCP, Poland) Grant No. 2023/49/N/ST6/02513. Simone Scardapane and Alessio Devoto are partially supported by Sapienza grant RG123188B3EF6A80 (CENTS).

We gratefully acknowledge Polish high-performance computing infrastructure PLGrid (HPC Center: ACK Cyfronet AGH) for providing computer facilities and support within computational grant no. PLG/2024/017202.

The contribution of Bartosz Wójcik to this research was conducted at the Faculty of Mathematics and Computer Science, and the Doctoral School of Exact and Natural Sciences of the Jagiellonian University.

References

Aguilar, G.; Ling, Y.; Zhang, Y.; Yao, B.; Fan, X.; and Guo, C. 2020. Knowledge distillation from internal representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 7350–7357.

Ardila, R.; Branson, M.; Davis, K.; Henretty, M.; Kohler, M.; Meyer, J.; Morais, R.; Saunders, L.; Tyers, F. M.; and Weber, G. 2020. Common Voice: A Massively-Multilingual Speech Corpus. *arXiv preprint at arXiv::1912.06670*.

Babu, A.; Wang, C.; Tjandra, A.; Lakhotia, K.; Xu, Q.; Goyal, N.; Singh, K.; von Platen, P.; Saraf, Y.; Pino, J.; Baevski, A.; Conneau, A.; and Auli, M. 2021. XLS-R: Self-supervised Cross-lingual Speech Representation Learning at Scale. *arXiv preprint at arXiv::2111.09296*.

Baevski, A.; Zhou, H.; Mohamed, A.; and Auli, M. 2020. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. *arXiv preprint at arXiv::2006.11477*.

Bengio, E.; Bacon, P.-L.; Pineau, J.; and Precup, D. 2015. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*.

Bengio, Y. 2013. Deep learning of representations: Looking forward. In *International conference on statistical language and speech processing*, 1–37. Springer.

Bolukbasi, T.; Wang, J.; Dekel, O.; and Saligrama, V. 2017. Adaptive neural networks for efficient inference. In *International Conference on Machine Learning*, 527–536. PMLR.

Cai, R.; Muralidharan, S.; Heinrich, G.; Yin, H.; Wang, Z.; Kautz, J.; and Molchanov, P. 2024. Flextron: Many-in-One Flexible Large Language Model. In *Forty-first International Conference on Machine Learning*.

Chen, Z.; Li, Y.; Bengio, S.; and Si, S. 2019. You look twice: Gaternet for dynamic filter selection in cnns. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 9172–9180.

Courbariaux, M.; Bengio, Y.; and David, J.-P. 2014. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*.

Dettmers, T.; Lewis, M.; Belkada, Y.; and Zettlemoyer, L. 2022. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*.

Fedus, W.; Dean, J.; and Zoph, B. 2022. A review of sparse expert models in deep learning. *arXiv preprint arXiv:2209.01667*.

Figurnov, M.; Collins, M. D.; Zhu, Y.; Zhang, L.; Huang, J.; Vetrov, D.; and Salakhutdinov, R. 2017. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1039–1048.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28.

Han, Y.; Huang, G.; Song, S.; Yang, L.; Wang, H.; and Wang, Y. 2021. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11): 7436–7456.

Haurum, J. B.; Escalera, S.; Taylor, G. W.; and Moeslund, T. B. 2023. Which Tokens to Use? Investigating Token Reduction in Vision Transformers. *arXiv preprint arXiv:2308.04657*.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hoefler, T.; Alistarh, D.; Ben-Nun, T.; Dryden, N.; and Peste, A. 2021. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22(1): 10882–11005.

Jain, G.; Hegde, N.; Kusupati, A.; Nagrani, A.; Buch, S.; Jain, P.; Arnab, A.; and Paul, S. 2024. Mixture of Nested Experts: Adaptive Processing of Visual Tokens. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Jang, E.; Gu, S.; and Poole, B. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

LeCun, Y.; Denker, J.; and Solla, S. 1989. Optimal brain damage. *Advances in neural information processing systems*, 2.

Li, C.; Wang, G.; Wang, B.; Liang, X.; Li, Z.; and Chang, X. 2021. Dynamic slimmable network. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, 8607–8617.

Lin, M.; Fu, J.; and Bengio, Y. 2019. Conditional computation for continual learning. *arXiv preprint arXiv:1906.06635*.

Meng, L.; Li, H.; Chen, B.-C.; Lan, S.; Wu, Z.; Jiang, Y.-G.; and Lim, S.-N. 2022. Adavit: Adaptive vision transformers for efficient image recognition. In *Proceedings of*

the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 12309–12318.

Patterson, D.; Gonzalez, J.; Hölzle, U.; Le, Q.; Liang, C.; Munguia, L.-M.; Rothchild, D.; So, D.; Texier, M.; and Dean, J. 2022. The Carbon Footprint of Machine Learning Training Will Plateau, Then Shrink. *arXiv preprint arXiv:2204.05149*.

Pugliese, R.; Regondi, S.; and Marini, R. 2021. Machine learning-based approach: Global trends, research directions, and regulatory standpoints. *Data Science and Management*.

Puigcerver, J.; Riquelme, C.; Mustafa, B.; and Houlsby, N. 2023. From Sparse to Soft Mixtures of Experts. *arXiv preprint arXiv:2308.00951*.

Qiu, Z.; Huang, Z.; and Fu, J. 2023. Emergent Mixture-of-Experts: Can Dense Pre-trained Transformers Benefit from Emergent Modular Structures? *arXiv preprint arXiv:2310.10908*.

Rao, Y.; Zhao, W.; Liu, B.; Lu, J.; Zhou, J.; and Hsieh, C.-J. 2021. Dynamicvit: Efficient vision transformers with dynamic token sparsification. *Advances in neural information processing systems*, 34: 13937–13949.

Riquelme, C.; Puigcerver, J.; Mustafa, B.; Neumann, M.; Jenatton, R.; Susano Pinto, A.; Keysers, D.; and Houlsby, N. 2021. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems*, 34: 8583–8595.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; Berg, A. C.; and Fei-Fei, L. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3): 211–252.

Scardapane, S.; Baiocchi, A.; Devoto, A.; Marsocci, V.; Minervini, P.; and Pomponi, J. 2024. Conditional computation in neural networks: Principles and research trends. *Intelligenza Artificiale*, 18(1): 175–190.

Scardapane, S.; Comminiello, D.; Scarpiniti, M.; Baccarelli, E.; and Uncini, A. 2020. Differentiable branching in deep networks for fast inference. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 4167–4171. IEEE.

Schwartz, R.; Dodge, J.; Smith, N. A.; and Etzioni, O. 2020. Green AI. *Commun. ACM*, 63(12): 54–63.

Shazeer, N.; Mirhoseini, A.; Maziarz, K.; Davis, A.; Le, Q.; Hinton, G.; and Dean, J. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Strubell, E.; Ganesh, A.; and McCallum, A. 2019. Energy and policy considerations for deep learning in NLP. *arXiv preprint arXiv:1906.02243*.

Tan, S.; Shen, Y.; Chen, Z.; Courville, A.; and Gan, C. 2023. Sparse Universal Transformer. *arXiv preprint arXiv:2310.07096*.

Teerapittayanon, S.; McDanel, B.; and Kung, H.-T. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*, 2464–2469. IEEE.

Tillet, P.; Kung, H.-T.; and Cox, D. 2019. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, 10–19.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Wołczyk, M.; Wójcik, B.; Bałazy, K.; Podolak, I. T.; Tabor, J.; Śmieja, M.; and Trzcinski, T. 2021. Zero time waste: Recycling predictions in early exit neural networks. *Advances in Neural Information Processing Systems*, 34: 2516–2528.

Wu, H.; Judd, P.; Zhang, X.; Isaev, M.; and Micikevicius, P. 2020. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602*.

Yin, H.; Vahdat, A.; Alvarez, J. M.; Mallya, A.; Kautz, J.; and Molchanov, P. 2022. A-vit: Adaptive tokens for efficient vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 10809–10818.

Yuksel, S. E.; Wilson, J. N.; and Gader, P. D. 2012. Twenty years of mixture of experts. *IEEE transactions on neural networks and learning systems*, 23(8): 1177–1193.

Zadouri, T.; Üstün, A.; Ahmadian, A.; Ermiş, B.; Locatelli, A.; and Hooker, S. 2023. Pushing Mixture of Experts to the Limit: Extremely Parameter Efficient MoE for Instruction Tuning. *arXiv preprint arXiv:2309.05444*.

Zagoruyko, S.; and Komodakis, N. 2016. Wide Residual Networks. In *British Machine Vision Conference 2016*. British Machine Vision Association.

Zhang, X.; Shen, Y.; Huang, Z.; Zhou, J.; Rong, W.; and Xiong, Z. 2022. Mixture of attention heads: Selecting attention heads per token. *arXiv preprint arXiv:2210.05144*.

Zhang, Z.; Lin, Y.; Liu, Z.; Li, P.; Sun, M.; and Zhou, J. 2021. Moefication: Transformer feed-forward layers are mixtures of experts. *arXiv preprint arXiv:2110.01786*.