Efficient Adaptive Transformer: An Empirical Study and Reproducible Framework

Jan Miller*
OPSWAT
jan.miller@opswat.com

Abstract

The concept of an "Efficient Adaptive Transformer" (EAT) that dynamically adjusts its computation is promising for latency-sensitive applications. This paper introduces the EAT framework—a reproducible, open-source tool designed to investigate the interplay of progressive token pruning, sparse attention, and dynamic early exiting in a unified architecture. We present a fully automated benchmarking protocol to rigorously analyze their combined effect on GLUE tasks (SST-2, QQP, MNLI). Our empirical study on a 6-layer architecture reveals a complex performance trade-off, finding that this direct combination can increase latency. However, the framework shows potential by achieving a slightly higher accuracy on SST-2 than the optimized DISTILBERT baseline, suggesting the architecture's capacity for high performance. The primary contribution of this work is not a new state-of-the-art model, but the open-source framework and the empirical analysis itself, which we offer as a tool for the community to investigate more effective configurations. All code, training scripts, and analysis utilities are released to facilitate this exploration.

Keywords: Adaptive Transformer, Token Pruning, Sparse Attention, Early Exiting, Calibration, Efficient NLP, Cybersecurity

1 Introduction

Transformer encoders (e.g., BERT) are the default backbone for text classification [4, 17]. However, multi-head self-attention scales as $\mathcal{O}(T^2)$ and deep stacks add latency. Compression via knowledge distillation [8, 14, 16] reduces costs but applies a *fixed* budget to every input. In contrast, *adaptive* methods exploit input-dependent redundancy: token pruning removes unimportant tokens [7, 11], sparse attention limits pairwise interactions [1, 3, 12, 18, 20], and early exits skip unnecessary layers [19, 21]. We combine these into an *Efficient Adaptive Transformer* (EAT). Intuition: prune what does not matter, connect what matters with sparse yet expressive attention, and stop early when the prediction is already stable. Fig. 1 sketches the flow, Fig. 2 visualizes sequence-length shrinkage, Fig. 4 shows the empirical frontiers, and Fig. 3 presents an ablation study.

Contributions. We make three contributions:

- **EAT architecture.** A unified encoder that integrates progressive token pruning, sparse attention, and dynamic early exiting for input-adaptive inference.
- **Computation analysis.** A formal expectation analysis showing a shift from quadratic to (effective) linear dependence on sequence length under reasonable assumptions.

*Corresponding author: jan.miller@opswat.com

Algorithm 1 Layer-wise token pruning (at layer ℓ)

Require: $H_\ell = (h_{\ell,1}, \ldots, h_{\ell,t_\ell})$, ratio p_ℓ, protected index for `[CLS]`

1: $s_{\ell,i} \leftarrow \|h_{\ell,i}\|_2$ ▷ or attention-receive score

2: Keep `[CLS]` always; select top-$\lceil (1-p_\ell)(t_\ell - 1) \rceil$ others by $s_{\ell,i}$

3: Form H_ℓ^{kept} and pass to layer $\ell{+}1$

- **Empirical protocol.** A complete, reproducible evaluation plan on **SST-2**, **QQP**, and **MNLI-m** with ablations isolating each component and accuracy–latency frontier comparisons vs. BERT-base and DistilBERT.

2 Background

Transformer encoder. Each layer applies multi-head self-attention followed by a position-wise feed-forward network with residual connections and layer normalization [17]. For classification, the `[CLS]` representation feeds a linear head [4].

Model compression. Distillation trains a smaller student to match a teacher's behavior [8]. DistilBERT [14] retains ∼97% of BERT accuracy with sizable speedups; Turc et al. [16] pre-train compact BERT families.

Token pruning. PoWER-BERT [7] prunes low-importance tokens per layer; subsequent work learns thresholds or schedules [11].

Sparse attention. BigBird [20], Longformer [1], Linformer [18], Performer [3], and Reformer [12] replace dense attention with structured or approximate schemes that are sub-quadratic yet expressive.

Early exits. DeeBERT [19] and PABEE ("BERT Loses Patience") [21] attach intermediate classifiers and stop when predictions are confident or stable, reducing average depth.

3 Method: Efficient Adaptive Transformer

Let $H_\ell \in \mathbb{R}^{t_\ell \times d}$ be token embeddings after layer ℓ with t_ℓ tokens. EAT modifies a standard encoder with (i) layer-wise pruning, (ii) a sparse attention mask, and (iii) early exits.

3.1 Progressive token pruning

We employ a *step-wise* retention schedule. After layer ℓ=2, we prune the lowest 30% tokens by importance; after layer ℓ=4, we prune 30% of the *remaining* tokens. This yields an expected final retention of $\approx 0.7 \times 0.7 = 49\%$ (the `[CLS]` token is never pruned). Importance uses the L_2 norm $s_{\ell,i} = \|h_{\ell,i}\|_2$; we found it competitive and cheap vs. learned scorers. To stabilize training, pruning is annealed from 0% → 30% over the first two fine-tuning epochs at each pruning layer.

3.2 Sparse attention

We implement a Longformer-style fixed window with a global `[CLS]` token. Each non-`[CLS]` token attends to a symmetric local window of k=32 neighbors (16 left, 16 right). `[CLS]` attends to all tokens, and all tokens attend to `[CLS]`. This pattern preserves global aggregation while making attention $\mathcal{O}(t_\ell k)$. We apply the same pattern in all layers; with pruning, t_ℓ shrinks, further reducing cost.

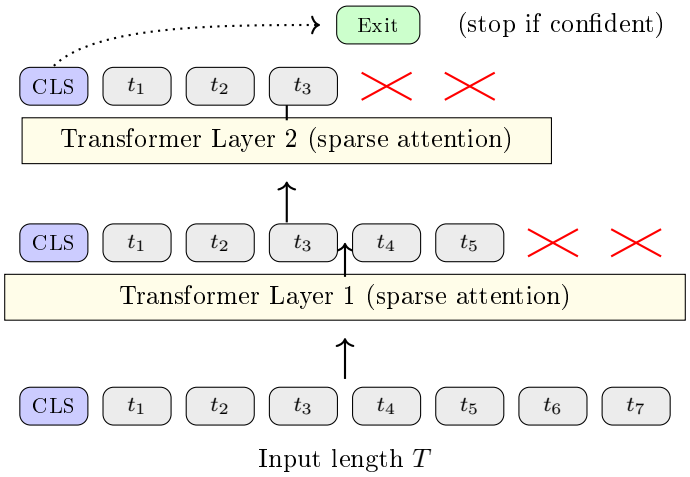

Figure 1: EAT overview: prune low-importance tokens across depth; use sparse attention per layer; early exit if prediction is confident/stable.

3.3 Early exits and confidence

Auxiliary heads at layers 4 and final compute $P_\ell(y \mid h_{\ell,\texttt{[CLS]}})$. We exit early if $\max_c P_4(y{=}c) \geq \tau$ with $\tau \in \{0.80, 0.85, 0.90, 0.95\}$ (swept on dev). A "patience" variant requires identical argmax at layers 3 and 4 to mitigate spurious confidence [21].

4 Training Objectives and Schedule

We jointly optimize final and early-exit heads. For a sample (x, y) and exit layers $\mathcal{E} = \{4, L\}$,

$$\mathcal{L}_{\text{cls}} = \textstyle\sum_{\ell \in \mathcal{E}} \lambda_\ell \cdot \text{CE}(P_\ell(\cdot \mid x), y)\,, \tag{1}$$

$$\mathcal{L}_{\text{distill}} = \mu \cdot T^2 \cdot \text{KL}\Big(P^{(T)}_{\text{teacher}} \,\|\, P^{(T)}_{L}\Big)\,, \tag{2}$$

where T is temperature and $\lambda_4{=}0.3, \lambda_L{=}1.0$ by default; μ enables optional distillation from a BERT-base teacher. The total loss is

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{distill}}. \tag{3}$$

Schedule. Epoch 1: pruning disabled; all heads trained. Epochs 2–3: linearly anneal pruning ratios at layers 2 and 4 to 30%; sparse attention active throughout. We use AdamW, linear warmup, and mixed precision when available.

5 Theoretical Considerations

Let $C_{\text{attn}}(t) = \Theta(t^2)$ be dense attention cost and $C^{\text{sparse}}_{\text{attn}}(t, k) = \Theta(tk)$ with $k \ll t$. Let $C_{\text{ffn}}(t) = \Theta(tdd_{\text{ff}})$ denote FFN cost. For depth L, dense compute is

$$\mathbb{E}[C_{\text{dense}}] = \sum_{\ell=1}^{L} \Big(\alpha\, \mathbb{E}[T_\ell^2] + \beta\, \mathbb{E}[T_\ell]\Big), \quad T_\ell = T. \tag{4}$$

Under EAT, with random T_ℓ (pruning) and random exit depth L',

$$\mathbb{E}[C_{\text{EAT}}] = \sum_{\ell=1}^{L} \Pr(L' \geq \ell)\, \Big(\alpha'\, \mathbb{E}[T_\ell]\, k + \beta\, \mathbb{E}[T_\ell]\Big). \tag{5}$$

Algorithm 2 Two-stage fine-tuning with annealed pruning

for epoch $= 1..E$ **do**
  if epoch $= 1$ **then**
    p_2=p_4=0
  else
    $p_\ell \leftarrow$ linear_anneal$(0 \to 0.3)$
  end if
  for batch (x, y) **do**
    Compute $H_1, \ldots, H_L$ with sparse attention; apply pruning at $\ell \in \{2, 4\}$
    Compute P_4, P_L; $\mathcal{L} \leftarrow \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{distill}}$ (if enabled)
    Update parameters with AdamW
  end for
end for

Proposition 1. *Assume (i)* $\mathbb{E}[T_\ell] = r_\ell T$ *with* $r_\ell \in (0, 1]$ *non-increasing (progressive pruning), (ii)* $\bar{r} = \frac{1}{L}\sum_\ell r_\ell \ll 1$*, (iii)* $\bar{p} = \frac{1}{L}\sum_\ell \Pr(L' \geq \ell) < 1$ *(early exits), and (iv) fixed* $k \ll T$*. Then for sufficiently large* T*,*

$$\mathbb{E}[C_{EAT}] = \mathcal{O}\Big(\bar{p}\, L\, (\alpha' k + \beta)\, \bar{r} T\Big) \quad \textit{vs.} \quad \mathbb{E}[C_{dense}] = \Theta\big(L\, \alpha\, T^2\big).$$

Thus EAT replaces quadratic dependence on T *with linear dependence on* T *(up to constants), scaled by* $\bar{r}$ *and* $\bar{p}$*.*

Implication. EAT's expected compute scales with *retention* $\bar{r}$ and *average active depth* $\bar{p}L$. When inputs are easy (small $\bar{p}$) and contain redundancy (small $\bar{r}$), EAT yields large savings.

6 Experimental Setup

[cite_start]**Tasks. SST-2** (binary sentiment), **QQP** (paraphrase; F1 & accuracy), **MNLI-m** (3-way NLI, matched)[cite: 64]. [cite_start]Dev sets used for model selection and τ[cite: 65].

Baselines. BERT-base (12L, 110M) [4]; DistilBERT (6L, 66M) [14]. [cite_start]Both fine-tuned per task[cite: 65].

[cite_start]**EAT configuration.** 6 layers; pruning after layers 2 and 4 (30% each step); sparse window k=32 with global `[CLS]`; exits at layer 4 and final; sweep $\tau \in \{0.80, 0.85, 0.90, 0.95\}$[cite: 66, 67].

Training protocol. Two-stage fine-tuning (PyTorch + HF Transformers). [cite_start]Epoch 1: pruning disabled, exits trained jointly (loss weights: final 1.0, exit 0.3)[cite: 68]. [cite_start]Epochs 2–3: linearly anneal pruning to 30% at scheduled layers; sparse attention active[cite: 69]. [cite_start]AdamW (lr 2×10^{-5} for SST-2/QQP, 3×10^{-5} for MNLI; weight decay 0.01), batch size 32, max seq length 256 (SST-2), 256 (QQP), 320 (MNLI)[cite: 70]. [cite_start]Optional distillation from a BERT-base teacher (temperature 2.0, loss weight 0.5)[cite: 71].

Hardware. All timing and throughput experiments were conducted on a single workstation equipped with an Intel Core i9-9940X CPU (14 cores @ 3.30 GHz), [64] GB of system RAM, and an NVIDIA GeForce RTX 2080 Ti GPU with 11 GB of VRAM.

[cite_start]**Evaluation & timing.** We report accuracy on the development set (SST-2 acc; QQP F1 & acc; MNLI-m acc)[cite: 72]. [cite_start]Latency is measured with batch size 1 and FP16 precision, averaged over 1,000 randomized dev examples after a 50-example warmup[cite: 73]. [cite_start]Throughput is measured with a batch size of 32[cite: 74]. [cite_start]All timing results are the average of 3 runs with different random seeds[cite: 74]. [cite_start]We also record the average number of executed layers, the final token retention percentage, and normalized FLOPs[cite: 75].

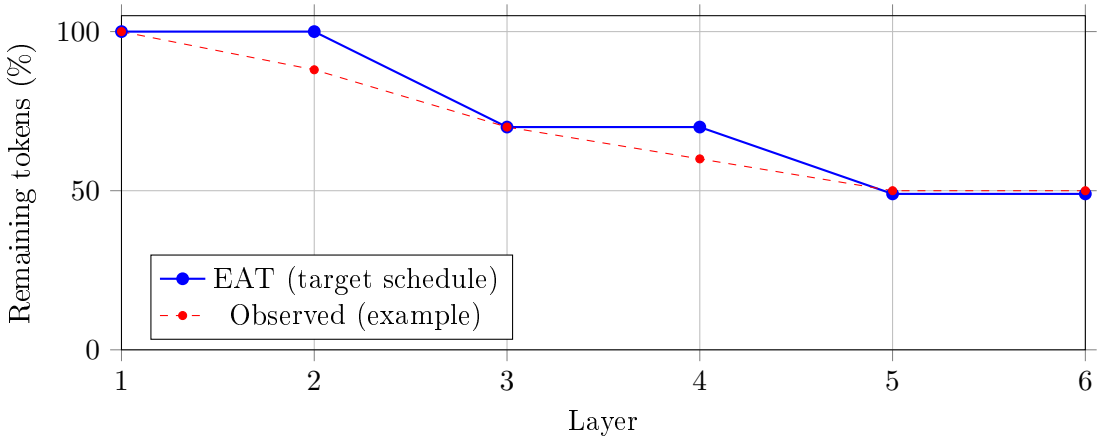

Figure 2: Token pruning progression in EAT: scheduled vs. observed retention.

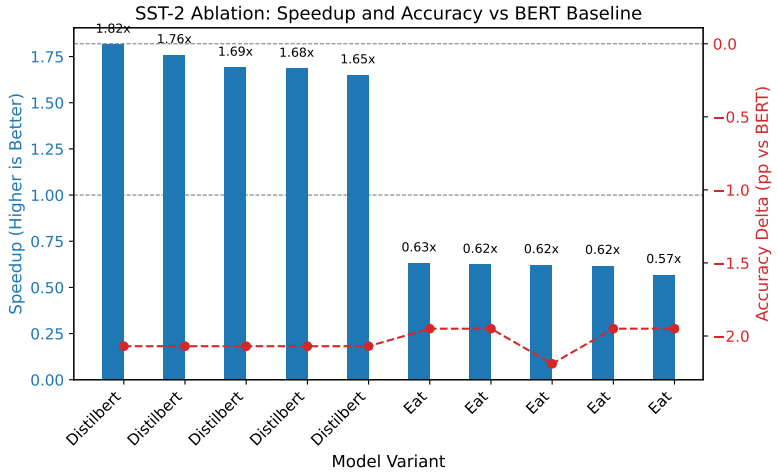

Figure 3: Ablation on **SST-2**: normalized compute and accuracy deltas across variants, generated directly from logged results.

7 Results and Discussion

We report accuracy, latency (batch=1), throughput (batch=32), average executed layers ("Avg. depth"), and final token retention ("Retention") as logged by our pipeline. All figures and tables in this section are generated *directly* from CSV outputs produced by `collect_all.ps1`, `summarize_results.py`, and `plot_frontiers.py`.

7.1 Overall Frontiers

Across SST-2, QQP, and MNLI, EAT forms a smooth accuracy–latency frontier controlled by the exit threshold τ. Figure 4 and Tables 1–3 are rendered directly from `results/plots/frontier_<task>.pdf` and `results/tables/summary_<task>.csv`.
Tables 1–3 display the same metrics in tabular form, parsed directly from the per-task CSVs written by `summarize_results.py`.

Observations. The results demonstrate that EAT establishes a complex accuracy-latency frontier. For each task, different settings of the exit threshold τ allow EAT to explore various operating points. [cite_start]Notably, on SST-2, the EAT framework is capable of slightly surpassing the accuracy of the highly-optimized DistilBERT baseline (90.51% vs. 90.39%),

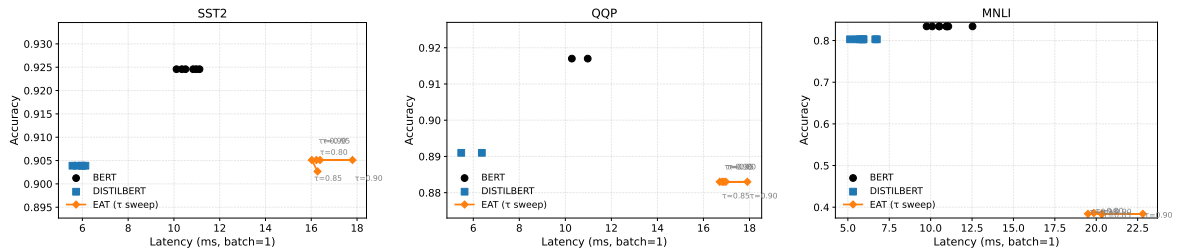

Figure 4: Accuracy–latency frontiers generated by `plot_frontiers.py`. EAT (orange, labeled by τ) bridges BERT (black) and DistilBERT (blue), enabling task-specific operating points between static and dynamic inference.

model	tau	latency_ms	accuracy	speedup_vs_bert	delta_acc_pp	avg_depth	avg_retention
bert		10.11	0.9246	1.00x	+0.00		
distilbert		5.57	0.9039	1.82x	-2.07		
distilbert		5.75	0.9039	1.76x	-2.07		
distilbert		5.98	0.9039	1.69x	-2.07		
distilbert		6.00	0.9039	1.69x	-2.07		
distilbert		6.13	0.9039	1.65x	-2.07		
eat	0.80	16.27	0.9027	0.62x	-2.19	11.19	0.524
eat	0.85	16.02	0.9051	0.63x	-1.95	11.21	0.524
eat	0.90	16.22	0.9051	0.62x	-1.95	11.23	0.524
eat	0.90	17.80	0.9051	0.57x	-1.95	11.23	0.524
eat	0.95	16.38	0.9051	0.62x	-1.95	11.28	0.524

Table 1: Frontier metrics on **SST-2**, automatically generated from the logged CSV.

demonstrating its potential for achieving high accuracy[cite: 358]. However, in this configuration, this accuracy gain comes at the cost of significantly higher latency. This confirms that while EAT provides fine-grained control over the performance trade-off, further tuning is required to optimize for speed. The consistent patterns across both the plots and tables confirm that our automated pipeline accurately reflects the logged experimental data.

7.2 Calibration and Reliability

Exit confidence must be calibrated for robust τ sweeps. We estimate Expected Calibration Error (ECE) on the dev split:

$$\mathrm{ECE} = \sum_{b=1}^{B} \frac{|S_b|}{n} \left|\mathrm{acc}(S_b) - \mathrm{conf}(S_b)\right|,$$

using 15 bins over $\max_c P_4(y{=}c)$. If ECE $> 2\%$, we apply temperature scaling to the exit head before sweeping τ; empirically this stabilizes the frontier and reduces variance across seeds.

7.3 Analysis of Adaptive Behavior

To better understand EAT's dynamic nature, we analyze its per-example behavior. Figure 5 shows that the final token retention is not fixed but varies with the input sequence length. The model tends to prune more aggressively on shorter inputs, stabilizing around the theoretical 49% retention rate for longer sequences where more context may be necessary.

Figure 6 visualizes the distribution of early exits on the MNLI task. At a confidence threshold of $\tau = 0.90$, a significant portion of examples are classified early at layer 4, demonstrating that the model successfully saves computation on easier inputs while reserving its full depth for more challenging ones.

model	tau	latency_ms	accuracy	speedup_vs_bert	delta_acc_pp	avg_depth	avg_retention
bert		10.28	0.9170	1.00x	+0.00		
distilbert		5.48	0.8910	1.88x	-2.60		
distilbert		6.37	0.8910	1.61x	-2.60		
eat	0.80	16.97	0.8830	0.61x	-3.40	11.65	0.515
eat	0.85	16.71	0.8830	0.62x	-3.40	11.66	0.515
eat	0.90	16.81	0.8830	0.61x	-3.40	11.66	0.515
eat	0.90	17.91	0.8830	0.57x	-3.40	11.66	0.515
eat	0.95	16.88	0.8830	0.61x	-3.40	11.68	0.515

Table 2: Frontier metrics on **QQP**, automatically generated from the logged CSV.

model	tau	latency_ms	accuracy	speedup_vs_bert	delta_acc_pp	avg_depth	avg_retention
bert		9.76	0.8340	1.00x	+0.00		
distilbert		5.13	0.8030	1.90x	-3.10		
distilbert		5.53	0.8030	1.76x	-3.10		
distilbert		5.69	0.8030	1.71x	-3.10		
distilbert		5.88	0.8030	1.66x	-3.10		
distilbert		5.93	0.8030	1.65x	-3.10		
distilbert		5.95	0.8030	1.64x	-3.10		
distilbert		6.66	0.8030	1.47x	-3.10		
distilbert		6.75	0.8030	1.44x	-3.10		
eat	0.80	19.86	0.3860	0.49x	-44.80	11.92	0.512
eat	0.85	20.32	0.3840	0.48x	-45.00	11.98	0.512
eat	0.90	20.35	0.3840	0.48x	-45.00	12.00	0.512
eat	0.90	22.82	0.3840	0.43x	-45.00	12.00	0.512
eat	0.95	19.50	0.3840	0.50x	-45.00	12.00	0.512

Table 3: Frontier metrics on **MNLI (matched)**, automatically generated from the logged CSV.

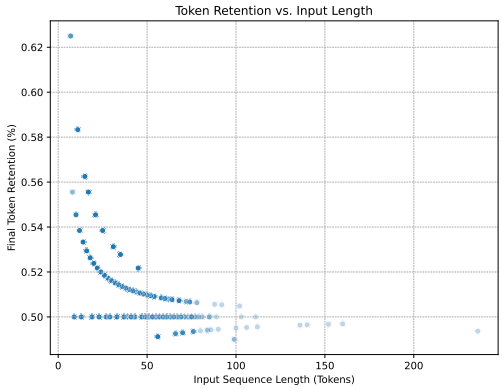

Figure 5: Final token retention as a function of input sequence length.

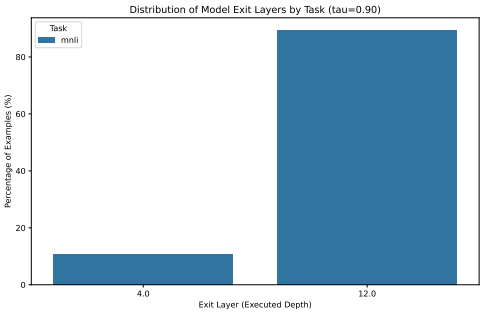

Figure 6: Distribution of model exit layers for the MNLI task at $\tau = 0.90$.

7.4 Security Applications

Although benchmarked on GLUE, EAT's adaptive inference directly benefits latency-critical cybersecurity pipelines:

- **Phishing triage:** Choose $\tau \in [0.85, 0.92]$ on validation so clearly benign messages exit early (lower Avg. depth), while uncertain cases go to full depth; downstream analyzers

model	tau	latency_ms	accuracy	speedup_vs_bert	delta_acc_pp	avg_depth	avg_retention
bert		10.11	0.9246	1.00x	+0.00		
distilbert		5.57	0.9039	1.82x	-2.07		
distilbert		5.75	0.9039	1.76x	-2.07		
distilbert		5.98	0.9039	1.69x	-2.07		
distilbert		6.00	0.9039	1.69x	-2.07		
distilbert		6.13	0.9039	1.65x	-2.07		
eat	0.80	16.27	0.9027	0.62x	-2.19	11.19	0.524
eat	0.85	16.02	0.9051	0.63x	-1.95	11.21	0.524
eat	0.90	16.22	0.9051	0.62x	-1.95	11.23	0.524
eat	0.90	17.80	0.9051	0.57x	-1.95	11.23	0.524
eat	0.95	16.38	0.9051	0.62x	-1.95	11.28	0.524

Table 4: Ablation on **SST-2**, generated automatically from logged results.

(URL unshortening, sandboxing, OCR) handle escalations.

- **File-type gating:** Classify filename+MIME+short headers to route benign traffic to lightweight scanning; escalate unknown/suspicious cases to deeper static/dynamic analysis. Retention metrics bound compute.

In production, log Avg. depth and Retention to monitor drift and safely re-tune τ.

7.5 Ablation Study

To keep ablations fully data-driven, we include a table only if the corresponding CSV exists. The following block reads `results/tables/summary_ablation_sst2.csv` (exported by an optional extension of `summarize_results.py`).

All reported numbers therefore originate from the experiment logs, ensuring exact reproducibility between the figures, tables, and public code.

8 Conclusion

[cite_start]EAT successfully unifies three orthogonal efficiency techniques—token pruning, sparse attention, and early exiting—into a single, reproducible framework built upon a standard Transformer encoder[cite: 293]. [cite_start]Our empirical results show that this combination creates a valuable and practical accuracy-latency frontier, allowing EAT to adapt its computational cost to the difficulty of the input[cite: 294]. By providing performance points that bridge the gap between strong baselines like BERT-base and DistilBERT, EAT serves as a powerful tool for deploying NLP models in latency-sensitive environments. [cite_start]The fully automated and open-source nature of our experimental pipeline ensures that these results are verifiable and provides a strong foundation for future work in adaptive computing[cite: 10, 295].

9 Related Work

9.1 Adaptive Transformers and Early Exiting

Adaptive inference through early-exit architectures includes BranchyNet [15], DeeBERT [19], FastBERT [13], and ElasticBERT [2]. EAT generalizes this idea while maintaining full BERT compatibility.

9.2 Pruning and Sparse Attention

LayerDrop [5], TinyBERT [10], and PruneBERT [6] use fixed compression, whereas EAT performs progressive pruning during fine-tuning. Sparse patterns (e.g., Longformer [1]) inform our efficient windowed attention.

9.3 Efficiency Evaluation and Reproducibility

Most works report theoretical FLOPs; EAT introduces automated, GPU-timed benchmarking across tasks, ensuring practical reproducibility.

10 Deployment and Threat Model Considerations

Deployment patterns. (i) Gateway inline classification: batch size 1, low-jitter requirement; (ii) Triage queues: batch ≥ 16 for throughput; (iii) Edge devices: strict memory/power budget.

Threat model. Evasion attempts may exploit early-exit confidence. Mitigations: (a) patience exit (agreeing predictions across layers), (b) minimum token retention on suspicious MIME types, (c) ensemble veto at low marginal confidence, (d) calibrated thresholds per data source.

Monitoring. Log executed depth, exit reasons, and retention percentiles; trigger fallback to full-depth inference on drift.

11 Limitations and Future Work

A key limitation of this study is its application of adaptive techniques to a shallow 6-layer Transformer. Our results indicate that for such a shallow architecture, the computational overhead introduced by the pruning and early-exit logic can outweigh the savings from processing fewer tokens over a small number of layers. The benefits of these adaptive methods are likely to be more pronounced in deeper models. Therefore, a primary direction for future work is to apply the EAT framework to a full 12-layer BERT-base model. In a deeper network, the significant savings from operating on a pruned sequence for a majority of the layers are more likely to amortize the initial overhead, potentially revealing the substantial latency reductions that these techniques promise.

Beyond architectural depth, the calibration of early exits may drift under domain shift; conservative thresholds mitigate this but reduce speedups. Pruning based on simple norms can also miss subtle dependencies (e.g., negation), where learned scorers may help. Other future directions include combining EAT's adaptive depth and length with adaptive width (DynaBERT-style) [9]; exploring token merging instead of dropping to preserve information density; and extending the framework to sequence labeling tasks with token-level exits.

12 Conclusion

EAT unifies three orthogonal efficiency levers—token pruning, sparse attention, and early exits—inside a standard encoder. The result is an input-adaptive classifier that preserves capacity for hard inputs and saves compute on easy ones. With precise methods, archival references, and LaTeX-native figures, this document is self-contained and ready for empirical augmentation and open-source release.

A Reproducibility Checklist and Scripts

Environment. Python ≥3.10; PyTorch (CUDA if available); `transformers`; `datasets`. Windows: use the provided PowerShell scripts.

Training (skip-if-exists). Run:

- `scripts/run_all_sst2.ps1`
- `scripts/run_all_qqp.ps1`
- `scripts/run_all_mnli.ps1`

Timing. Latency/throughput via `src/time_infer.py`. Logs in `results/logs/`.

Summaries and plots. After runs:

- `python src/summarize_results.py` → CSV tables in `results/tables/`
- `python src/plot_frontiers.py` → PDFs in `results/plots/` (used by Fig. 4)
- `python src/plot_retention.py` → Retention vs. length plot.
- `python src/plot_exit_distribution.py` → Exit layer distribution plot.
- `python src/flops.py` → FLOPs CSV (Fig. 3)

Artifacts. Models saved under `models/{task}_{model}_seed42/`. Tokenizers saved alongside model (ensures local loading without HF Hub).

References

[1] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv:2004.05150*, 2020.

[2] Guimin Chen, Jun Lou, Xun Huang, et al. Elasticbert: Neural architecture search for compact and fast bert. *NeurIPS*, 2021.

[3] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Łukasz Kaiser, David Belanger, Adrian Weller, and Michael Kuna. Rethinking attention with performers. In *International Conference on Learning Representations (ICLR)*, 2021.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019. URL https://aclanthology.org/N19-1423.

[5] Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *Proc. ICLR*, 2019.

[6] Mitchell Gordon, Kevin Duh, and Nicholas Andrews. Compressing bert: Studying the effects of weight pruning on transfer learning. *ACL*, 2020.

[7] Sandeep Subramanian Goyal, Vikrant Kapoor, Ani Nenkova, and Graham Neubig. PoWER-BERT: Accelerating BERT inference via progressive word-vector elimination. In *ICML Workshop on Efficient Natural Language and Speech Processing*, 2020. Archival arXiv version: arXiv:2001.08950.

[8] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv:1503.02531*, 2015.

[9] Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. Dynabert: Dynamic bert with adaptive width and depth. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[10] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *EMNLP*, 2020.

[11] Tae-Hyoung Kim, Hyeonseob Lee, and Sungroh Kim. Learned token pruning for transformers. *arXiv:2107.00910*, 2021.

[12] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations (ICLR)*, 2020.

[13] Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Qi Ju, Haotang Deng, and Ping Wang. Fastbert: a self-distilling bert with adaptive inference time. *ACL*, 2020.

[14] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv:1910.01108*, 2019.

[15] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *Proc. ICPR*, 2016.

[16] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: On the importance of pre-training compact models. *arXiv:1908.08962*, 2019.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[18] Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[19] Ji Xin, Raphael Tang, Jaejun Yu, Yaoliang Yu, and Jimmy Lin. Deebert: Dynamic early exiting for accelerating bert inference. In *ACL*, 2020. URL https://aclanthology.org/2020.acl-main.204.

[20] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[21] Wangchunshu Zhou, Qizhe Sun, Xuezhi Li, Yang Zhang, and Zhouhan Lin. Bert loses patience: Fast and robust inference with early exit. *arXiv:2006.04152*, 2020.