
Efficient PLMs
from the perspective of Early Exiting

Preliminaries

Datasets	CoNLL 2003		SST-2		MNLI		
Model	Time	Score	Time	Score	Time	Score	Overall Score
$\text{BERT}_{\text{BASE}}$	2.68	3.18	2.70	3.13	2.67	3.19	9.5
$\text{BERT}_{\text{LARGE}}$	8.51	1.00	8.46	1.00	8.53	1.00	3.00
$\text{XLNet}_{\text{BASE}}$	5.16	1.65	5.01	1.69	5.10	1.67	5.01
$\text{XLNet}_{\text{LARGE}}$	14.84	0.57	14.69	0.58	15.27	0.56	1.71
$\text{RoBERTa}_{\text{BASE}}$	2.65	3.21	2.68	3.16	2.70	3.16	9.53
$\text{RoBERTa}_{\text{LARGE}}$	8.35	1.02	8.36	1.01	8.70	0.98	3.01
$\text{ALBERT}_{\text{BASE}}$	2.65	3.21	2.68	3.16	2.72	3.14	9.51
$\text{ALBERT}_{\text{LARGE}}$	8.49	1.00	8.44	1.00	8.78	0.97	2.97

Table 4: Multi-task Baseline Inference Costs. Time is given in milliseconds and score is computed by the division of $\text{Time}_{\text{BERT}_{\text{LARGE}}}/\text{Time}_{\text{model}}$. The experiments are conducted on a single RTX 2080 Ti GPU following the evaluation criterion similar to the fine-tuning part. The inference time between tasks is more consistent compared to the fine-tuning phase.

What & Why?

To exit after being *confident* [1] to the results during inference.

Don’t need to execute **ALL** the model layers.

w/ EE

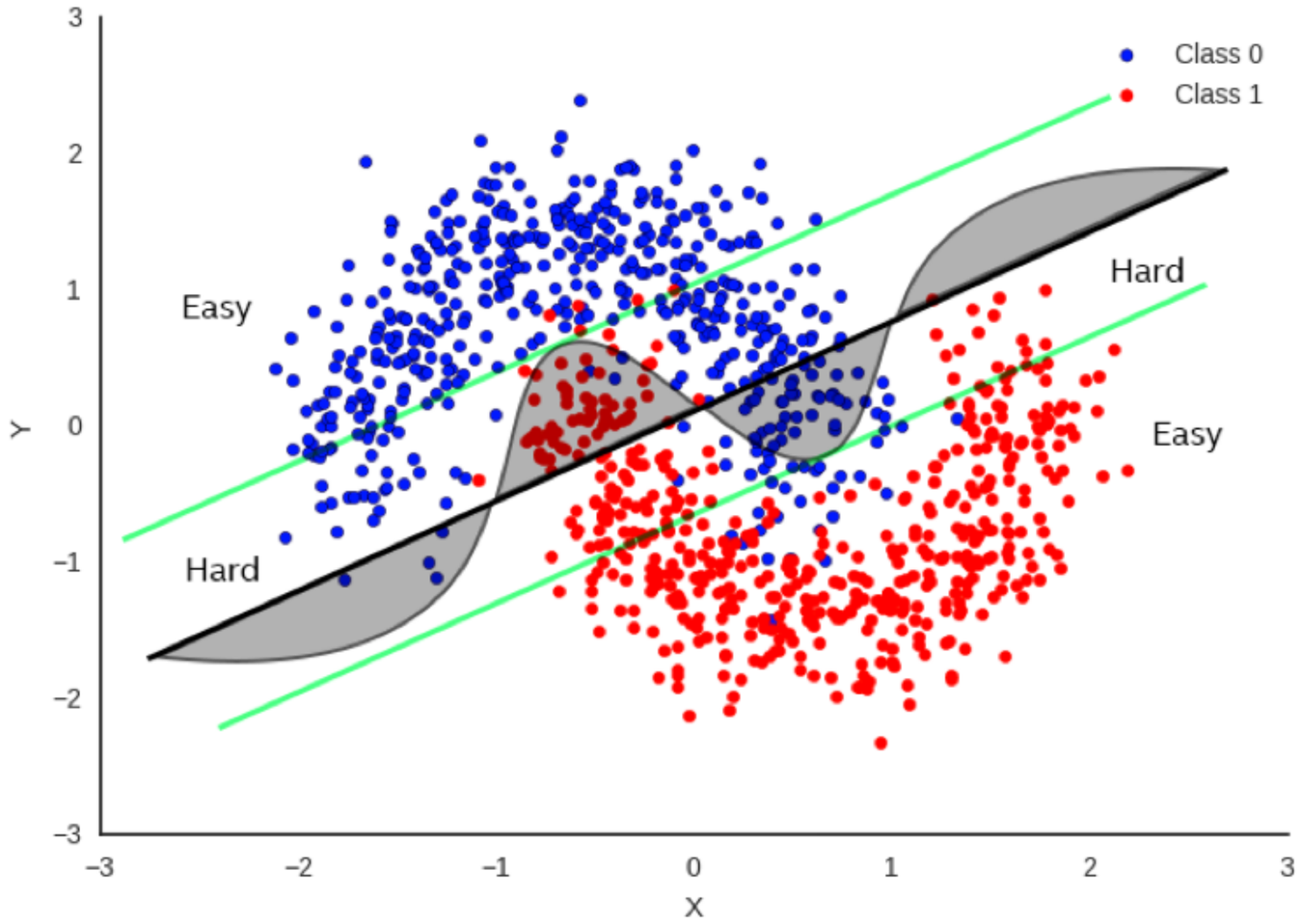
Layer 1 → Layer 2 → Layer 3 → Layer 4 → … → Layer 12

X → X → √ → **Output**

Depending on inputs

1. confidence ≠ perplexity

Illustrated Premise



- *Easy* inputs

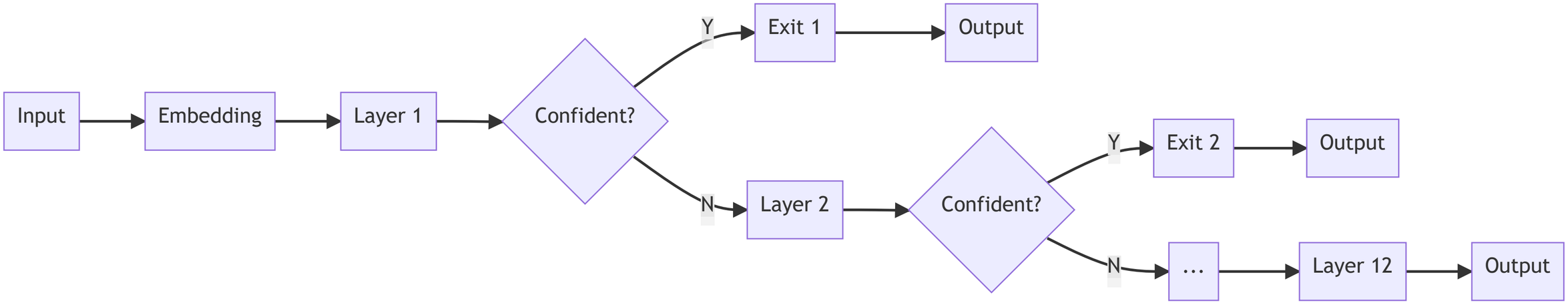
  confident, aggressive
- *Hard* inputs

  confused, conservative

- Measuring difficulty?

  open question

Illustrated Architecture



Branchy

Pros & Cons

- Pros
  1. **Elastic** - single model with dynamic configs at inference time
  2. **Efficiency** - reduced inference time & energy consumption
  3. **Effectiveness** - reduced overthinking (*overfitting* of inference) [1]
- Cons (Insights for new papers)
  1. **Trade-off** - performance V.S. time-saving
  2. **Compatibility** - no uniformed confidence evaluation for different tasks
  3. **Implementation** - training approach influences performance

1. Shallow-Deep Networks: Understanding and Mitigating Network Overthinking [ICML2019] University of Maryland

How?

1. Entropy [1]
2. Patience [2]
3. Learning-based
4. Pre-training [3]

---

1. DeeBERT: Dynamic Early Exiting for Accelerating BERT Inference [ACL 2020]
   University of Waterloo, Vector Institute of AI
2. BERT Loses Patience: Fast and Robust Inference with Early Exit [NIPS 2020]
   Beihang University, University of California, MSRA

Method 1 — Entropy

$$H(x) = -\sum_i p(i) \ln p(i) = \ln\left(\sum_{i=1}^{n} e^{x_i}\right) - \frac{\sum_{i=1}^{n} x_i e^{x_i}}{\sum_{i=1}^{n} e^{x_i}}$$

- $H(x) < E_T$? [1]

  entropy lower than threshold → confident → exit

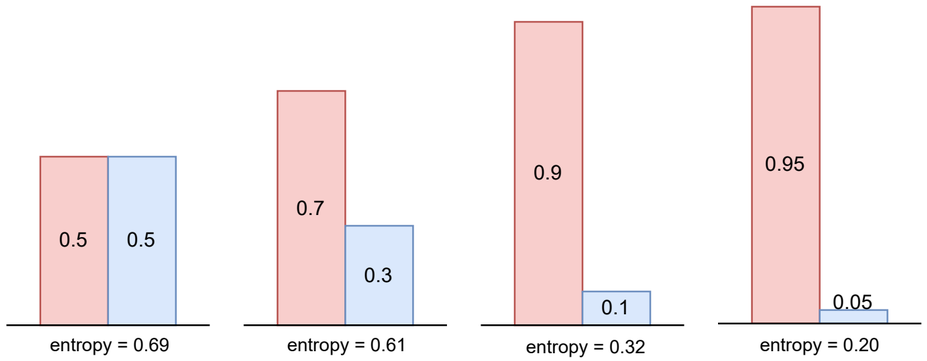
  otherwise, higher than threshold → confused → run more layers

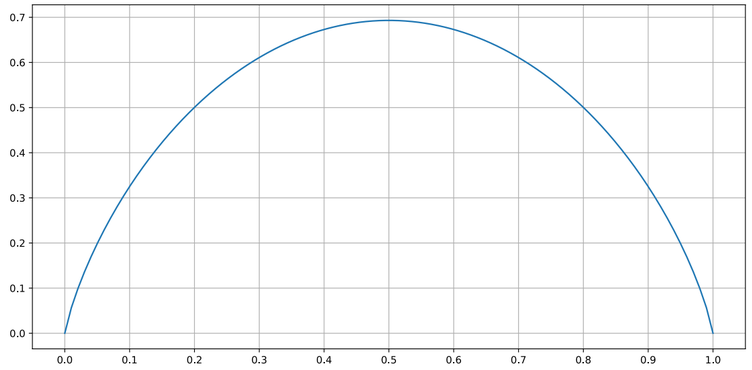
1. DeeBERT: Dynamic Early Exiting for Accelerating BERT Inference [ACL 2020]
   University of Waterloo, Vector Institute of AI

Illustrated Entropy — Binary Class

$$H(x) = F(p_0) = -\big(p_0 \ln p_0 + (1 - p_0) \ln(1 - p_0)\big)$$

where $p_0, p_1 = \text{softmax}(x)$, and $p_1 = 1 - p_0$





- $\arg\max_{p_0} F(p_0) = 0.5, \quad \max F(p_0) = \ln 2 = 0.69$
- $\lim_{p_0 \to 0^+} F(p_0) = 0 \quad , \quad \lim_{p_0 \to 1^-} F(p_0) = 0$

Training DeeBERT

Two-stage fine-tuning

1. Ordinary fine-tuning
   - Embedding, all Transformers, and the last classifier
2. Each classifier (ex. the last) training
   - Freeze the above parameters that are already fine-tuned
   - $\mathcal{L} = \sum_{i=1}^{n-1} \mathcal{L}_i$

Time-consuming training

Implements

```python
def entropy(x):
    # x: torch.Tensor, logits BEFORE softmax
    exp_x = torch.exp(x)
    A = torch.sum(exp_x, dim=1)    # sum of exp(x_i)
    B = torch.sum(x*exp_x, dim=1)  # sum of x_i * exp(x_i)
    return torch.log(A) - B/A
```

```
# GLUE BertForSequenceClassification pooled_outputs
# x.size() = (batch_size=1, num_labels=2)
[[0.0, 0.1]]
# NER  BertForTokenClassification sequence_outputs
# x.size() = (batch_size=1, seq_len=128, num_labels=9)
[[[0.0, 0.1, ..., 0.8],
  [1.0, 1.2, ..., 1.8],
             ...      ]]
```

Fix

```
def entropy(x):
    # x: torch.Tensor, logits BEFORE softmax
    # softmax normalized prob distribution
    x = torch.softmax(x, dim=-1)
    # entropy calculation on probs: -\sum(p \ln(p))
    return -torch.sum(x*torch.log(x), dim=-1)
```

- Sometimes `dim=1`, always `dim=-1`
- `torch.softmax()` more efficient than manual calculation

DeeBERT Results — BERT

	SST-2		MRPC		QNLI		RTE		QQP		MNLI-(m/mm)	
	Acc	Time	F1	Time	Acc	Time	Acc	Time	F1	Time	Acc	Time
	BERT-base											
Baseline	93.6	36.72s	88.2	34.77s	91.0	111.44s	69.9	61.26s	71.4	145min	83.9/83.0	202.84s
DistilBERT	-1.4	-40%	-1.1	-40%	-2.6	-40%	-9.4	-40%	-1.1	-40%	-4.5	-40%
DeeBERT	-0.2	-21%	-0.3	-14%	-0.1	-15%	-0.4	-9%	-0.0	-24%	-0.0/-0.1	-14%
DeeBERT	-0.6	-40%	-1.3	-31%	-0.7	-29%	-0.6	-11%	-0.1	-39%	-0.8/-0.7	-25%
DeeBERT	-2.1	-47%	-3.0	-44%	-3.1	-44%	-3.2	-33%	-2.0	-49%	-3.9/-3.8	-37%

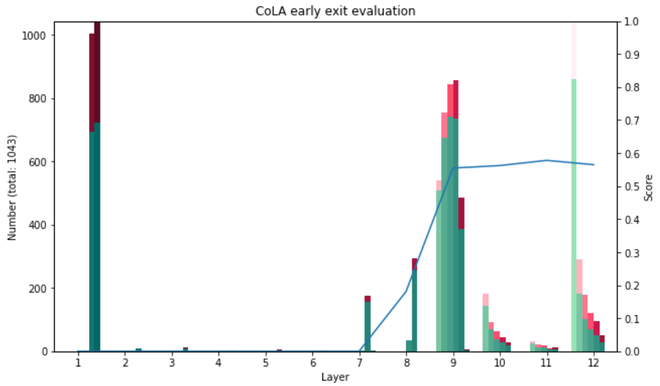
DeeBERT Results — RoBERTa

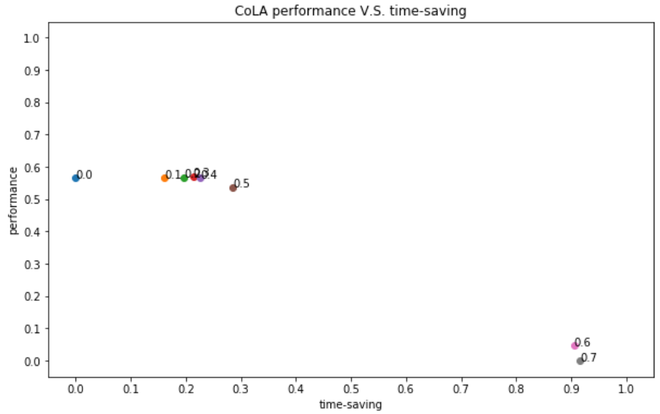
	SST-2		MRPC		QNLI		RTE		QQP		MNLI-(m/mm)	
	Acc	Time	F1	Time	Acc	Time	Acc	Time	F1	Time	Acc	Time
	RoBERTa-base											
Baseline	94.3	36.73s	90.4	35.24s	92.4	112.96s	67.5	60.14s	71.8	152min	87.0/86.3	198.52s
LayerDrop	-1.8	-50%	-	-	-	-	-	-	-	-	-4.1	-50%
DeeBERT	**+0.1**	**-26%**	**+0.1**	**-25%**	-0.1	-25%	-0.6	-32%	**+0.1**	**-32%**	**-0.0/-0.0**	**-19%**
DeeBERT	**-0.0**	**-33%**	**+0.2**	**-28%**	-0.5	-30%	-0.4	-33%	**-0.0**	**-39%**	-0.1/-0.3	-23%
DeeBERT	-1.8	-44%	-1.1	-38%	-2.5	-39%	-1.1	-35%	-0.6	-44%	-3.9/-4.1	-29%

Reduced Overthinking? Why RoBERTa, not BERT?

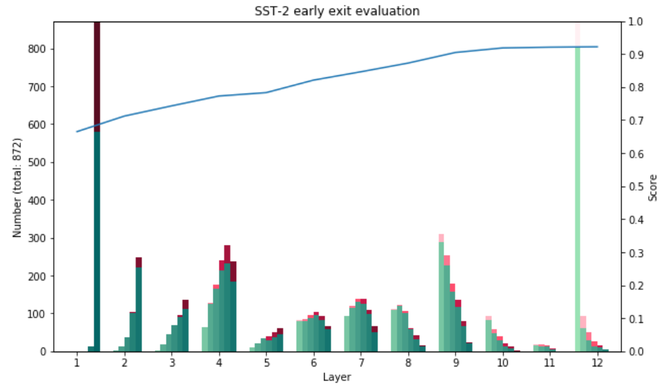
Illustrated Results — Single-Sentence

CoLA*



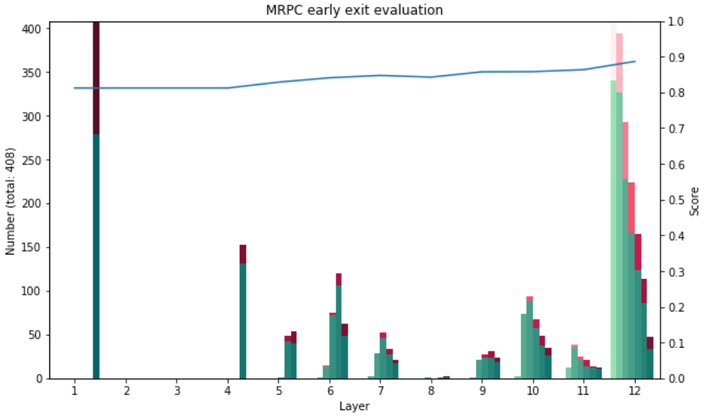


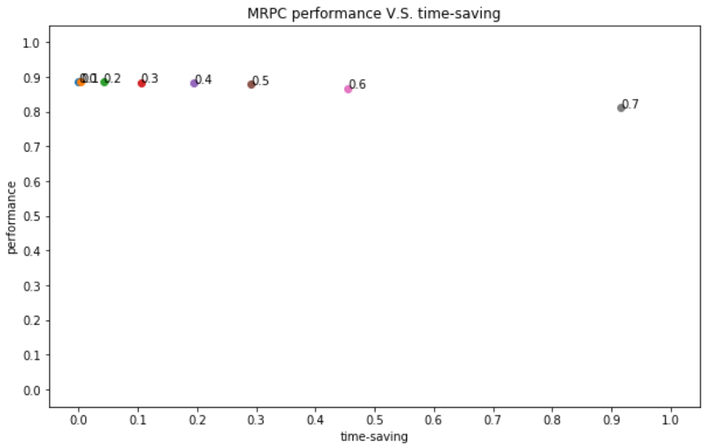
SST-2



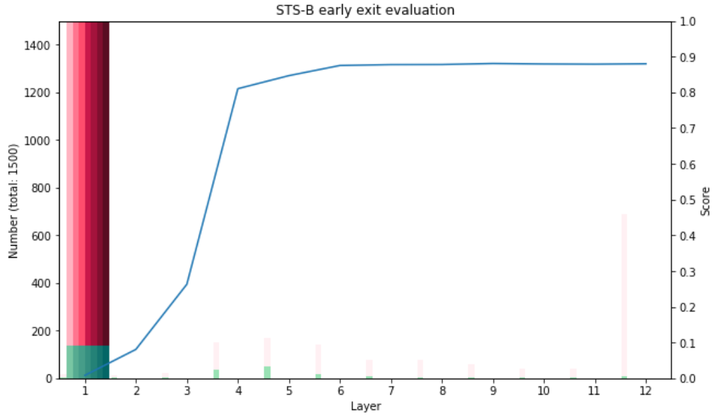
Illustrated Results — Similarity and Paraphrase

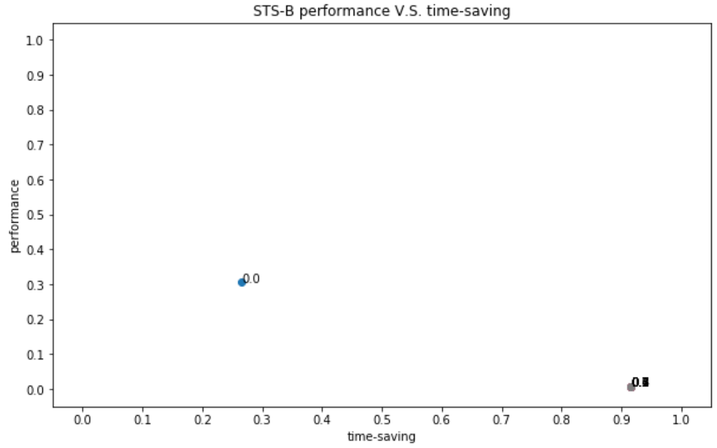
MRPC



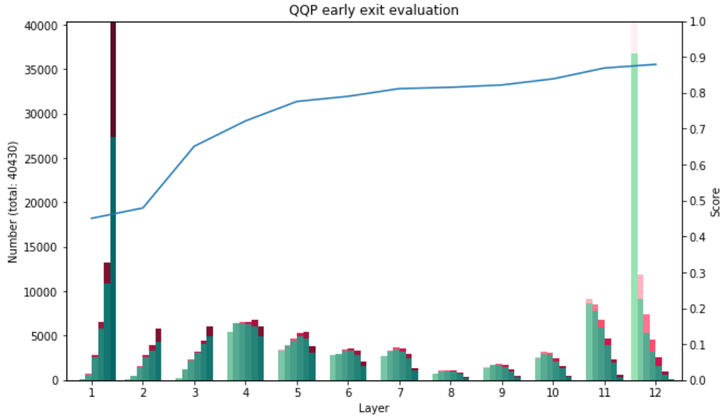


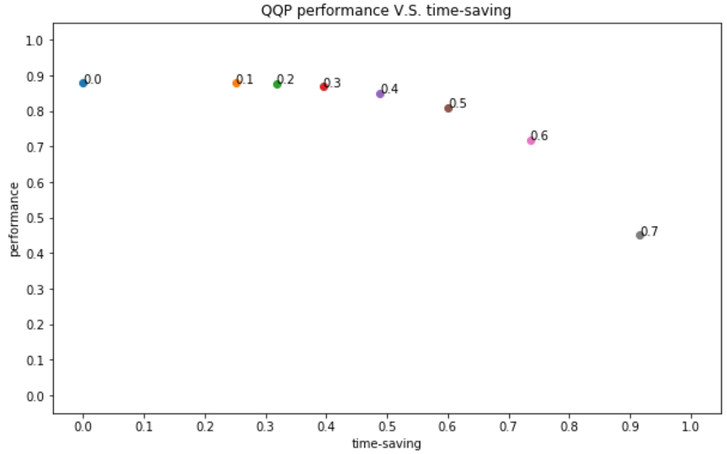
STS-B*





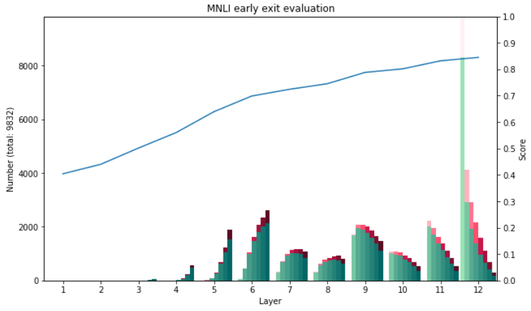
QQP

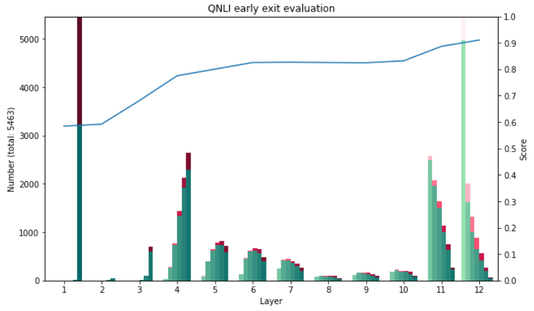


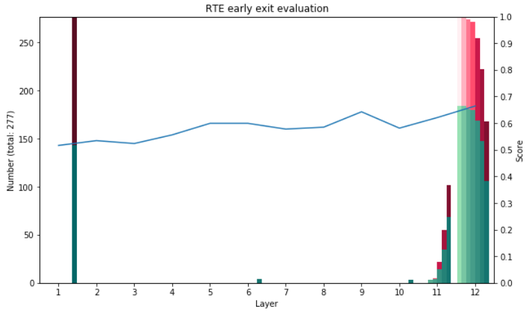


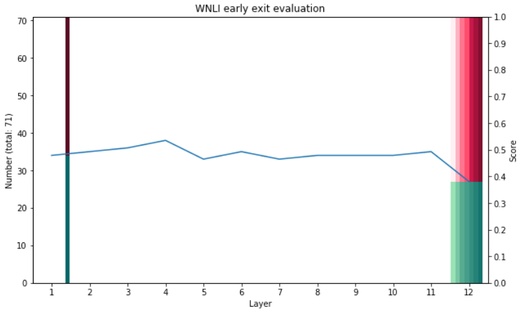
Illustrated Results — Inference

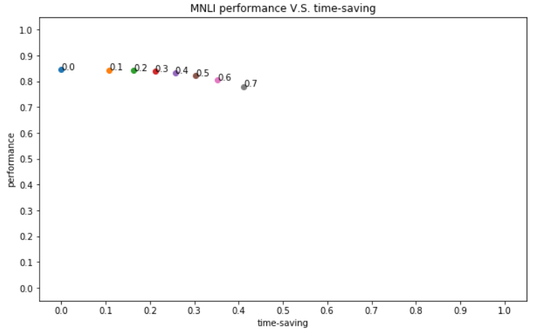
MNLI | QNLI | RTE | WNLI*

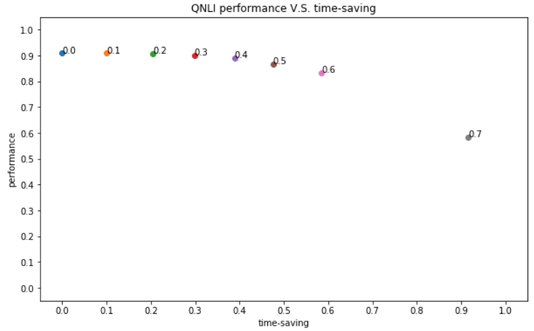


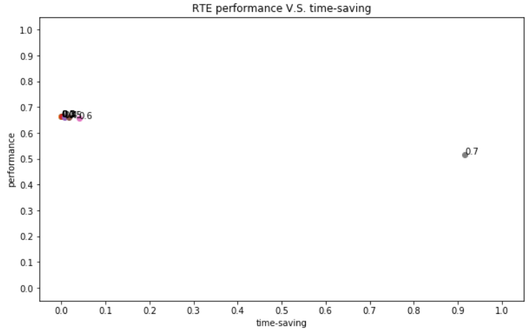


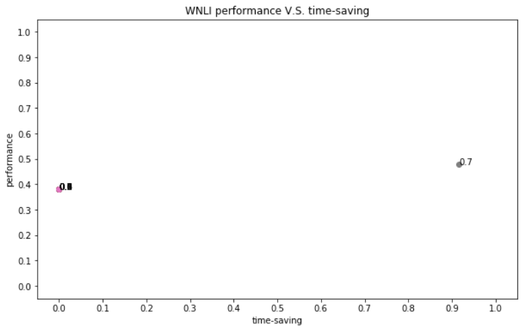






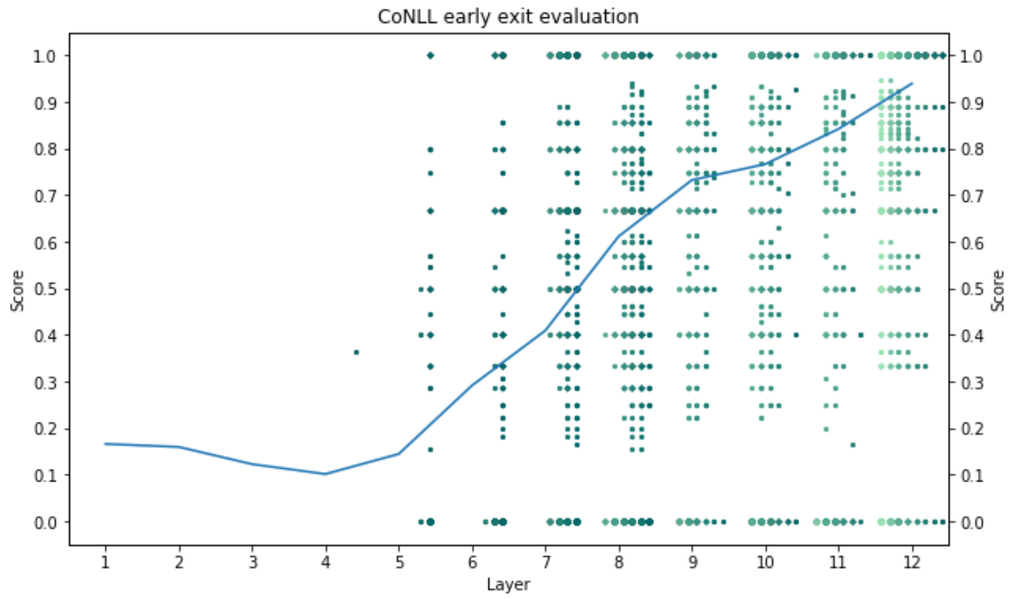


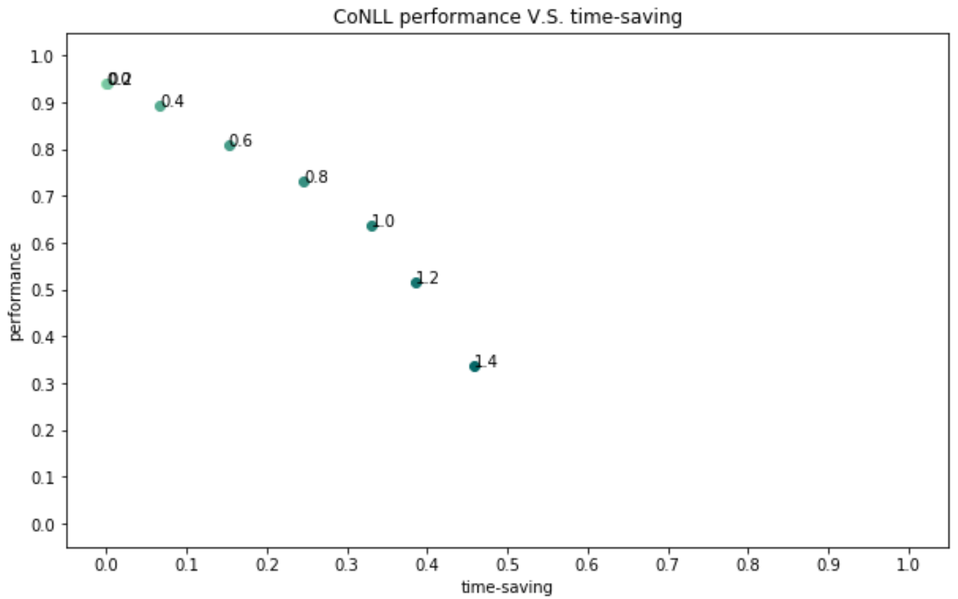




Extending

- NER adaption on CoNLL-2003 dataset (Token Classification, 9 labels)
  1. [CLS] token entropy → mean value of non-padding token entropies
  2. maximum threshold: $\ln 9 = 2.20$





Q: What are EASY Inputs?

> The movie was awesome.

Positive

> You are good at messing up.

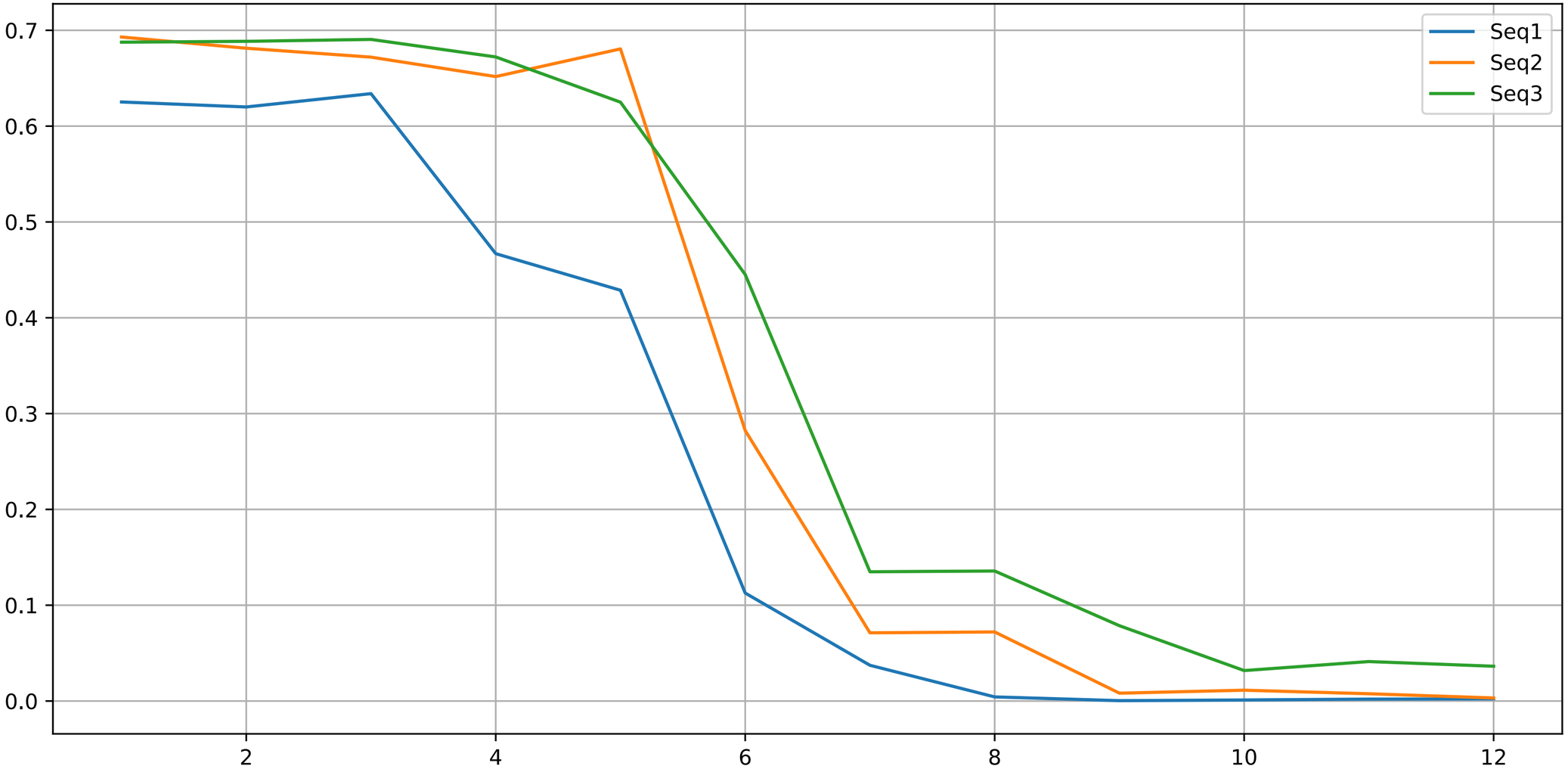
Negative

> I wonder whether the plot was written by a 12-year-old or by an award-winning writer.

Negative

Layer	1	2	3	4	5	6	7	8	9	10	11	12
Seq 1 +	√	√	√	√	√	√	√	√	√	√	√	√
	0.6252	0.6200	0.6339	0.4669	0.4287	0.1127	0.0373	0.0043	0.0004	0.0011	0.0020	0.0023
Seq 2 -	√	√	√	×	×	×	×	×	×	×	×	×
	0.6930	0.6813	0.6720	0.6517	0.6806	0.2823	0.0712	0.0721	0.0082	0.0113	0.0076	0.0032
Seq 3 -	×	√	×	√	√	√	√	√	√	√	√	√
	0.6876	0.6885	0.6905	0.6722	0.6250	0.4451	0.1349	0.1357	0.0786	0.0318	0.0412	0.0363

Illustrated Inputs



Inputs Difficulty

Spearman's correlation ρ between *confidence* and input features [1]

```
confidences, predictions = torch.max(softmaxes, dim=1)
```

Dataset	Length	Consistency
AG	0.13	0.37
IMDB	−0.17	0.47
SST	−0.19	0.36
SNLI	−0.08	0.44
MNLI	−0.13	0.39

1. Length - confidence of 1st classifier

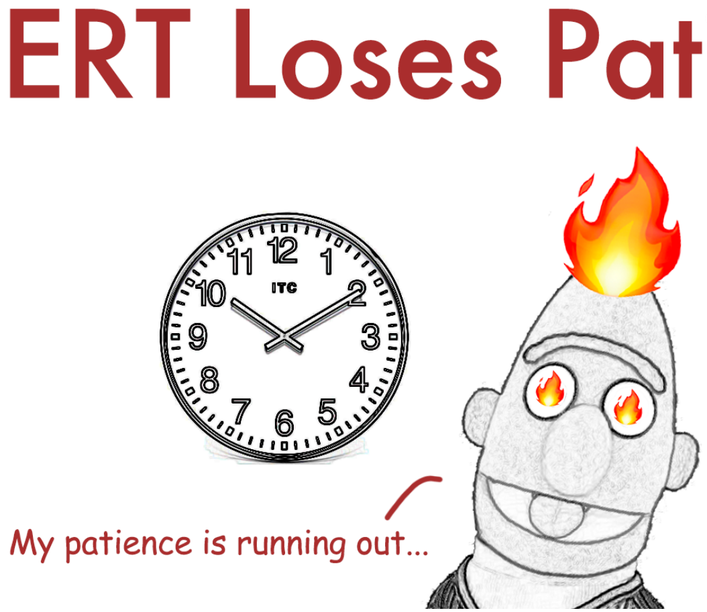
   Loose, negative (exc. AG, news topic detect)

2. Consistency - confidence of 1st classifier

   Medium, positive

1. The Right Tool for the Job: Matching Model and Instance Complexities [ACL 2020]
   Allen AI, University of Washington

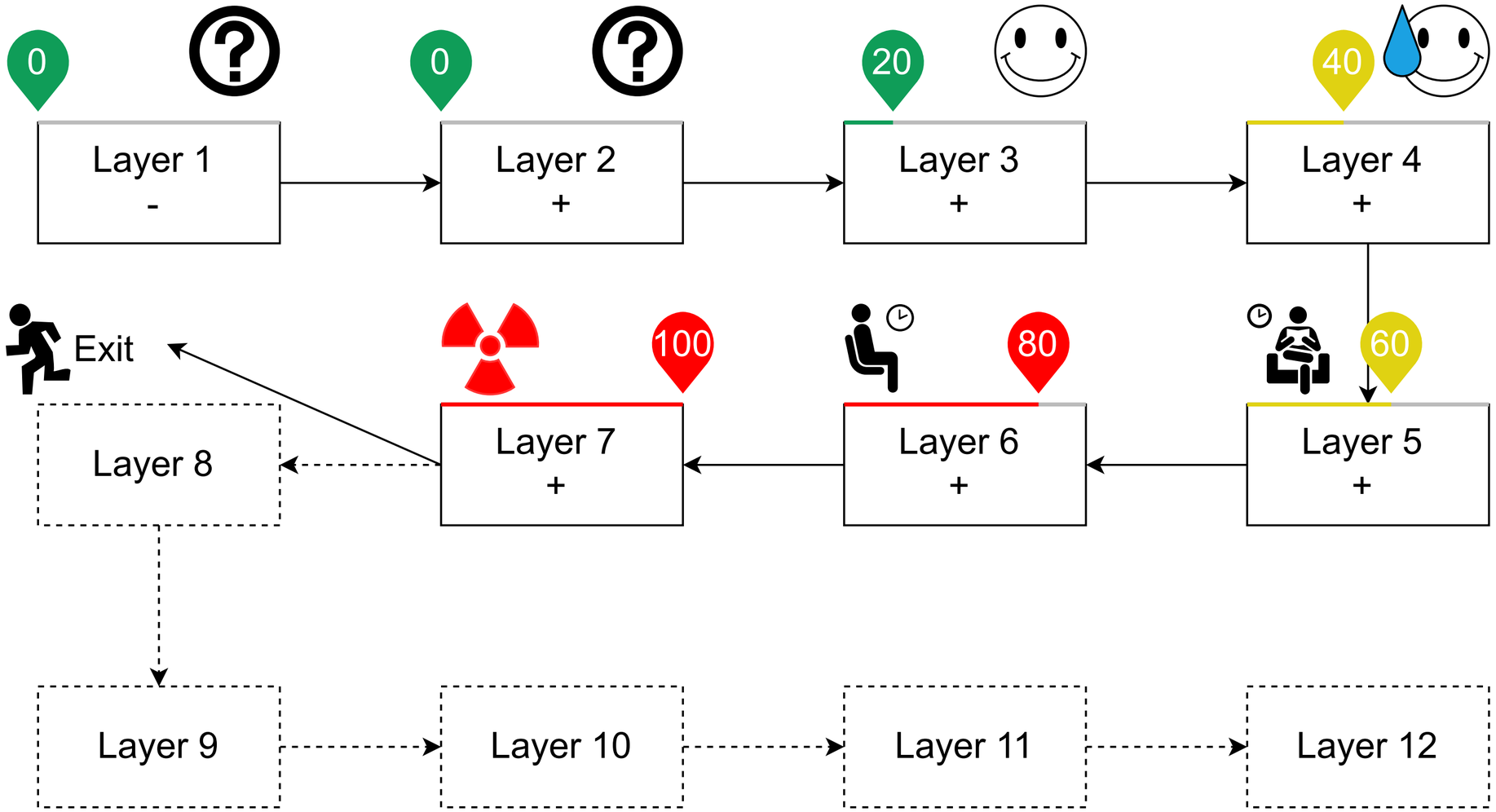
Method 2 — Patience



PABEE [1]

1. BERT Loses Patience: Fast and Robust Inference with Early Exit [NIPS 2020]
   Beihang University, University of California, MSRA

Illustrated Patience



Training PABEE

One-stage fine-tuning

- Weighted average loss following [1] $\mathcal{L} = \frac{\sum_{j=1}^{n} j \cdot \mathcal{L}_j}{\sum_{j=1}^{n} j}$

what about descending [2] $\mathcal{L}_{desc} = \frac{\sum_{j=1}^{n} (n+1-j) \cdot \mathcal{L}_j}{\sum_{j=1}^{n} j}$

---

1. Shallow-Deep Networks: Understanding and Mitigating Network Overthinking [ICML 2019] University of Maryland
2. 模型早退技术(二): 中间分类层训练方法 [知乎]

PABEE Results — BERT

Table 2: Experimental results (median of 5 runs) of BERT based models on the development set of GLUE. We mark "-" on STS-B for BranchyNet and Shallow-Deep since they do not support regression.

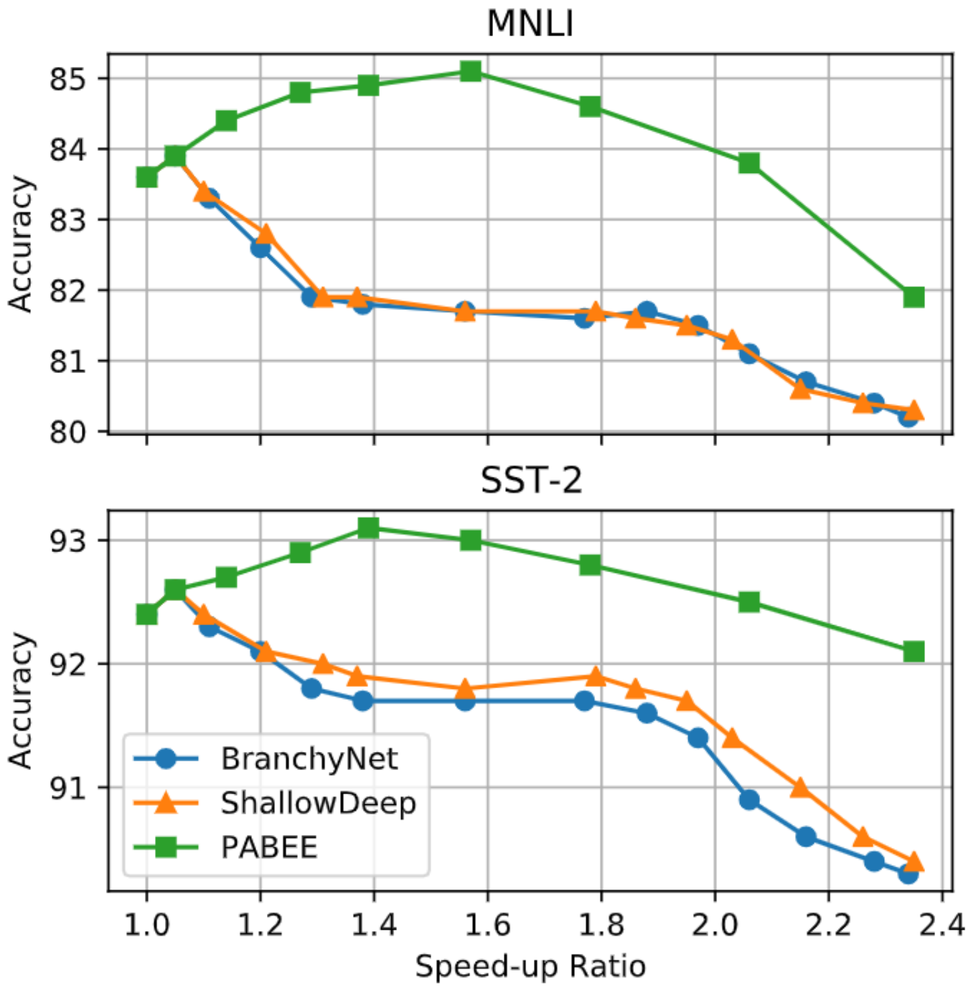
Method	**#Param**	**Speed -up**	**MNLI** (393K)	**SST-2** (67K)	**STS-B** (5.7K)
BERT-base [1]	108M	1.00×	84.5	92.1	88.9
BERT-6L	66M	1.96×	80.1	89.6	81.2
BERT-9L	87M	1.30×	81.4	90.5	85.0
DistilBERT [17]	66M	1.96×	79.0	90.7	81.2
BERT-PKD [23]	66M	1.96×	81.3	91.3	86.2
BERT-of-Theseus [23]	66M	1.96×	82.3	91.5	**88.7**
BranchyNet [26]	108M	1.87×	80.3	90.4	-
Shallow-Deep [5]	108M	1.91×	80.5	90.6	-
PABEE *(ours)*	108M	1.62×	**83.6**	**92.0**	**88.7**

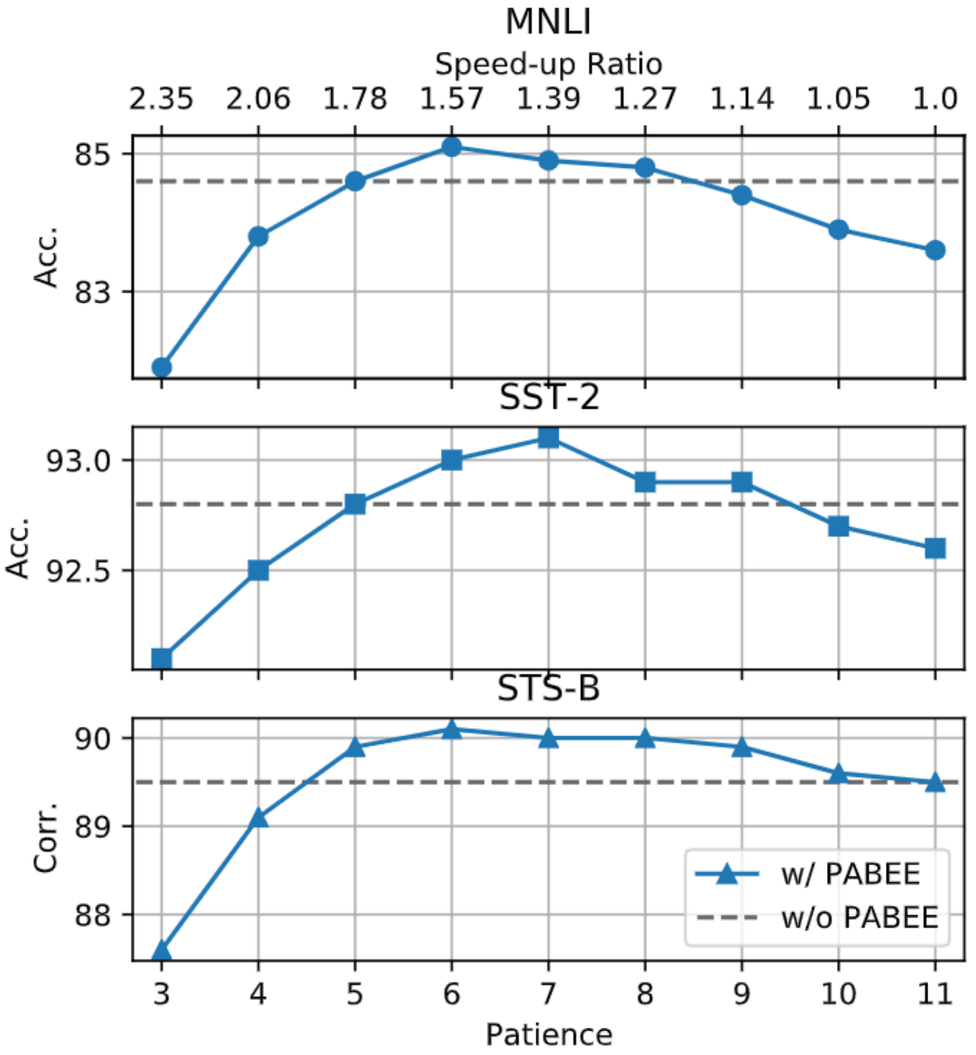
PABEE Results — ALBERT

Table 1: Experimental results (median of 5 runs) of models with ALBERT backbone on the development set and the test set of GLUE. The numbers under each dataset indicate the number of training samples. The acceleration ratio is averaged across 8 tasks. We mark "-" on STS-B for BranchyNet and Shallow-Deep since they do not support regression.

Method	**#Param**	**Speed -up**	**CoLA** (8.5K)	**MNLI** (393K)	**MRPC** (3.7K)	**QNLI** (105K)	**QQP** (364K)	**RTE** (2.5K)	**SST-2** (67K)	**STS-B** (5.7K)	**Macro Score**
Dev. Set											
ALBERT-base [4]	12M	1.00×	58.9	84.6	89.5	91.7	89.6	78.6	92.8	89.5	84.4
ALBERT-6L	12M	1.96×	53.4	80.2	85.8	87.2	86.8	73.6	89.8	83.4	80.0
ALBERT-9L	12M	1.30×	55.2	81.2	87.1	88.7	88.3	75.9	91.3	87.1	81.9
LayerDrop [22]	12M	1.96×	53.6	79.8	85.9	87.0	87.3	74.3	90.7	86.5	80.6
HeadPrune [20]	12M	1.22×	54.1	80.3	86.2	86.8	88.0	75.1	90.5	87.4	81.1
BranchyNet [26]	12M	1.88×	55.2	81.7	87.2	88.9	87.4	75.4	91.6	-	-
Shallow-Deep [5]	12M	1.95×	55.5	81.5	87.1	89.2	87.8	75.2	91.7	-	-
PABEE *(ours)*	12M	1.57×	**61.2**	**85.1**	**90.0**	**91.8**	**89.6**	**80.1**	**93.0**	**90.1**	**85.1**
Test Set											
ALBERT-base [4]	12M	1.00×	54.1	84.3	87.0	90.8	71.1	76.4	94.1	85.5	80.4
PABEE *(ours)*	12M	1.57×	**55.7**	**84.8**	**87.4**	**91.0**	**71.2**	**77.3**	**94.1**	**85.7**	**80.9**

PABEE Results — Overthinking





Theorem

- PABEE **improves** the accuracy $s.t.$ $n - t < \left(\frac{1}{2q}\right)^t \left(\frac{p}{q}\right) - p$ where
  - t - patience
  - n - *internal* classifiers (IC) ($n = 12$) ???
  - q - error rate of internal classifiers (ex. the final)
  - p - error rate of the final classifier and the original classifier (w/o ICs) ???

1. main text $n - t < \left(\frac{1}{2q}\right)^t \left(\frac{p}{q}\right) - p$
2. appendix recap $n - t < \left(\frac{1}{2q}\right)^{t+1} p - q$
3. appendix proof $n - t < \frac{\left(\frac{1}{2q}\right)^t \left(\frac{p}{q}\right) - q}{1-q} < \left(\frac{1}{2q}\right)^t \left(\frac{p}{q}\right) - q$???

Method 3 — Pre-training

ElasticBERT [1] — Pre-trained multi-exit Transformer

> Q: Why don’t fine-tune BERT, but pre-train a new one?

> A: Gap between pre-training and fine-tuning hurt the performance!

1. Towards Efficient NLP: A Standard Evaluation and A Strong Baseline [WIP]
   Fudan University, Huawei Poisson Lab

Training ElasticBERT — Objectives

Pre-training objectives

- masked language mode (MLM)
- sentence order prediction (SOP)

$$\mathcal{L} = \sum_{l=1}^{L} (\mathcal{L}_l^{\text{MLM}} + \mathcal{L}_l^{\text{SOP}})$$

∵ The two losses are applied to each layer of the model

∴ Number of layers can be flexibly scaled

Training ElasticBERT — Gradient Equilibrium

Re-scaling (re-normalizing) gradient [1]

Summed loss → overlap of subnetworks → gradient imbalance

Averaged loss → small gradients → hinders convergence

$$\nabla_{w_i}^{(\mathrm{GE})} \mathcal{L}_j = \prod_{i \le h < j} \frac{k-h}{k-h+1} \times \frac{1}{k-j+1} \times \nabla_{w_i} \mathcal{L}_j$$

$$= \frac{1}{k-i+1} \nabla_{w_i} \mathcal{L}_j$$

1. Improved Techniques for Training Adaptive Deep Networks [ICCV 2019]
   THU, Baidu, University of Oxford

Training ElasticBERT — Grouped Training

Summing losses at all layers → slow pre-training + increased memory

1. Divide L exits into G groups
2. Optimize losses of exit classifiers within each group
3. Round Robin between different batches

$$\mathcal{L} = \sum_{l \in \mathcal{G}_i} \left(\mathcal{L}_l^{\text{MLM}} + \mathcal{L}_l^{\text{SOP}} \right)$$

E.g., for 12 Layers, $\mathcal{G}_1 = \{1, 3, 5, 7, 9, 11, 12\}$, $\mathcal{G}_2 = \{2, 4, 6, 8, 10, 12\}$

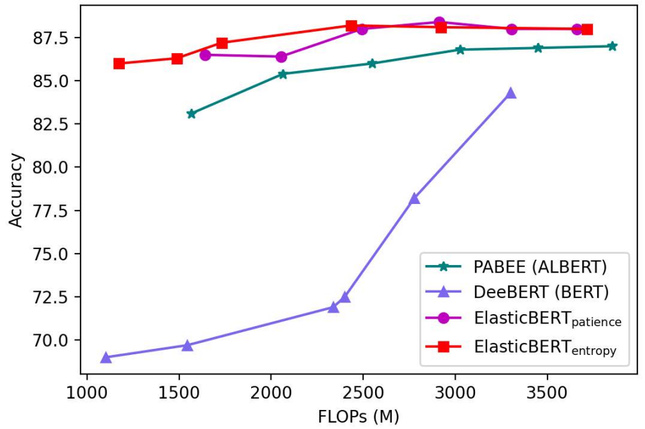
ElasticBERT Results — Static Base

Models	#Params	CoLA	MNLI-(m/mm)	MRPC	QNLI	QQP	RTE	SST-2	STS-B	Average
$\text{BERT}_{\text{BASE}}$	109M	56.5	84.6/84.9	87.6	91.2	89.6	69.0	92.9	89.4	82.9
$\text{ALBERT}_{\text{BASE}}$	12M	56.8	84.9/85.6	90.5	91.4	89.2	78.3	92.8	90.7	84.5
$\text{RoBERTa}_{\text{BASE}}$	125M	63.6	87.5/87.2	90.8	92.7	90.3	77.5	94.8	90.9	86.1
$\text{ElasticBERT}_{\text{BASE}}$	109M	64.3	85.3/85.9	91.0	92.0	90.2	76.5	94.3	90.7	85.6
$\text{BERT}_{\text{BASE}}$-6L	67M	44.6	81.4/81.4	84.9	87.4	88.7	65.7	90.9	88.1	79.2
$\text{ALBERT}_{\text{BASE}}$-6L	12M	52.4	82.6/82.2	89.0	89.8	88.7	70.4	90.8	89.6	81.7
$\text{RoBERTa}_{\text{BASE}}$-6L	82M	44.4	84.2/84.6	87.9	90.5	89.8	60.6	92.1	89.0	80.3
MobileBERT	25M	52.1	83.9/83.5	89.3	91.3	88.9	63.5	91.3	87.2	81.2
TinyBERT-6L	67M	46.3	83.6/83.8	88.7	90.6	89.1	73.6	92.0	89.4	81.9
$\text{ElasticBERT}_{\text{BASE}}$-6L	67M	**53.7**	**84.3**/84.2	**89.7**	90.8	89.7	**74.0**	**92.7**	**90.2**	**83.3**
Test Set Results										
TinyBERT-6L	67M	42.5	83.2/82.4	86.2	89.6	79.6	73.0	91.8	85.7	79.3
$\text{ElasticBERT}_{\text{BASE}}$-6L	67M	**49.1**	**83.7/83.4**	**87.3**	**90.4**	**79.7**	68.7	**92.9**	**86.9**	**80.3**

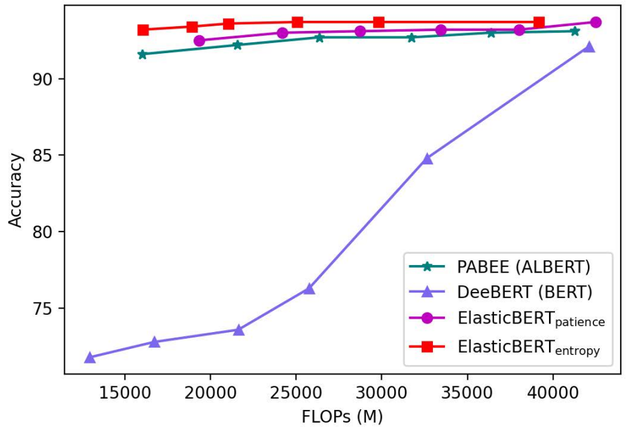
ElasticBERT Results — Static Large

Models	#Params	CoLA	MNLI-(m/mm)	MRPC	QNLI	QQP	RTE	SST-2	STS-B	Average
$\text{BERT}_{\text{LARGE}}$	335M	61.6	86.2/86	90.1	92.2	90.1	72.9	93.5	90.4	84.8
$\text{ALBERT}_{\text{LARGE}}$	18M	60.1	86/86.1	90.4	91.6	89.6	83.0	95.2	91.4	85.9
$\text{RoBERTa}_{\text{LARGE}}$	355M	66.4	89/89.6	91.6	94.2	90.7	86.6	95.4	92.3	88.4
$\text{ElasticBERT}_{\text{LARGE}}$	335M	66.3	88/88.5	92.0	93.6	90.9	83.1	95.3	91.7	87.7
$\text{BERT}_{\text{LARGE}}$-12L	184M	42.6	81/81.1	81.6	87.2	89.3	65.7	89.3	88.7	78.5
$\text{ALBERT}_{\text{LARGE}}$-12L	18M	59.0	85.3/85.8	90.1	91.4	89.6	76.7	93.3	91.3	84.7
$\text{RoBERTa}_{\text{LARGE}}$-12L	204M	62.3	86.3/86.2	89.4	92.3	90.4	71.8	93.5	91.1	84.8
$\text{ElasticBERT}_{\text{LARGE}}$-12L	184M	62.1	86.2/**86.4**	89.5	**92.5**	**90.6**	**79.1**	93.0	**91.6**	**85.7**
Test Set Results										
$\text{RoBERTa}_{\text{LARGE}}$-12L	204M	59.4	86.4/85.2	87.6	91.6	80.4	67.3	94.6	89.5	82.4
$\text{ElasticBERT}_{\text{LARGE}}$-12L	184M	57.0	85.4/84.9	**87.7**	**92.3**	**81.2**	**71.8**	92.9	**89.7**	**82.6**

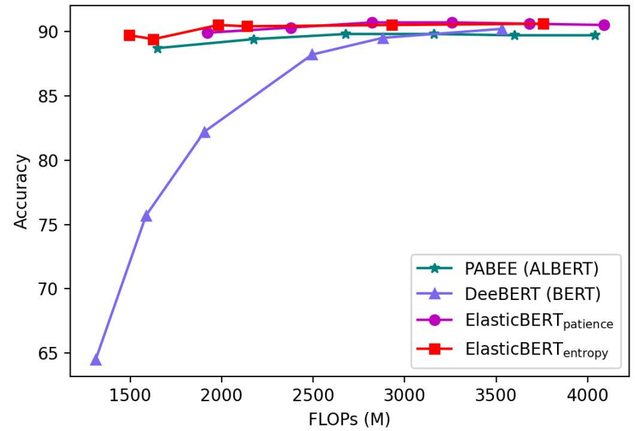
ElasticBERT Results — Dynamic



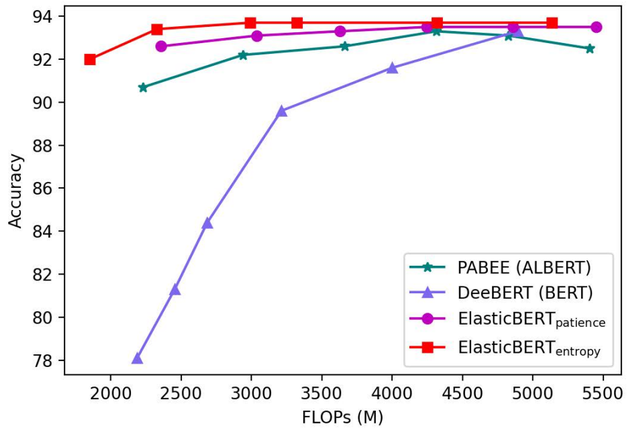
SST-2



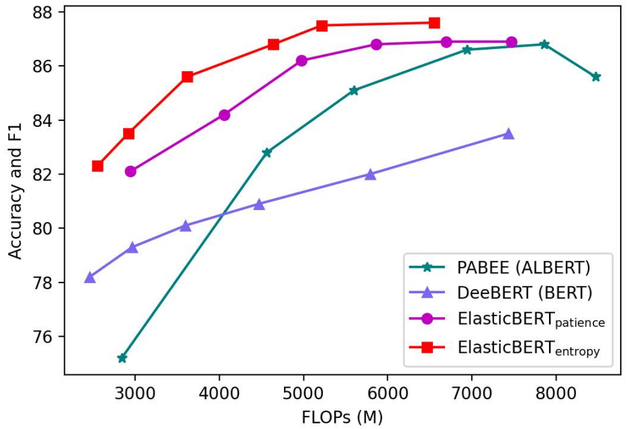
IMDb



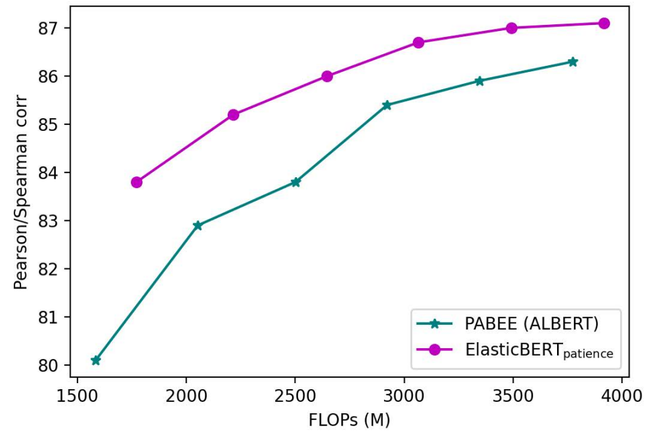
SNLI



SciTail



MRPC



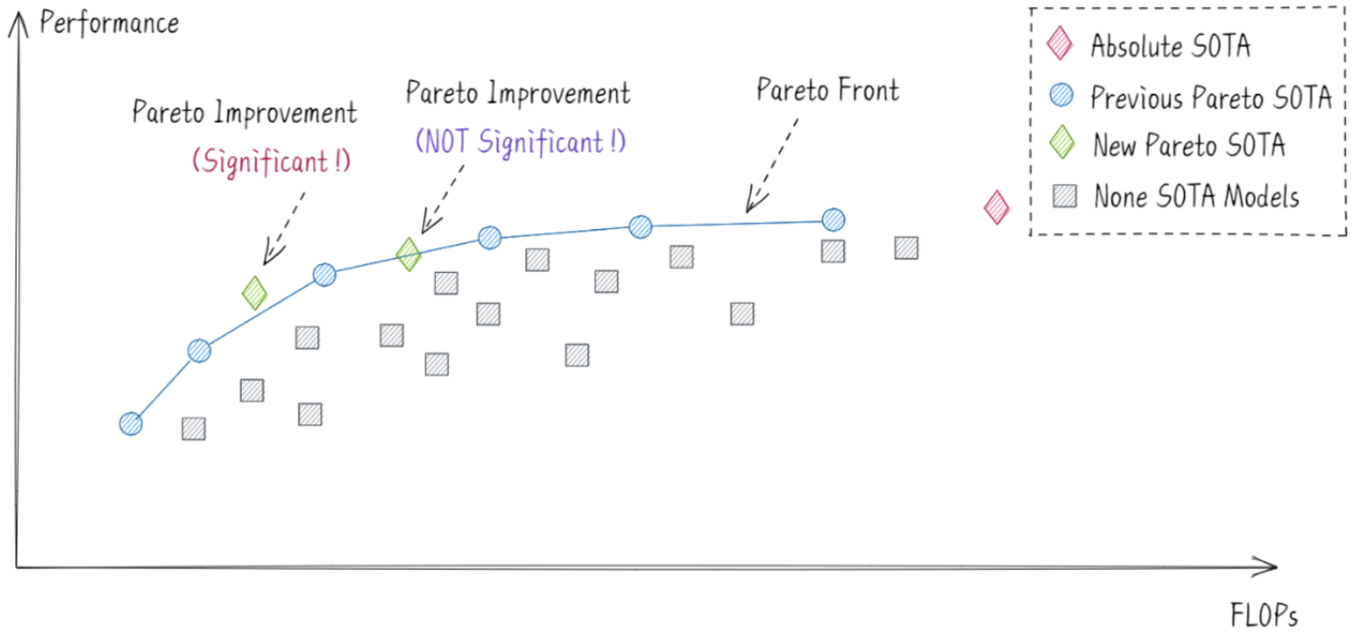
STS-B

Benchmarking

SOTA

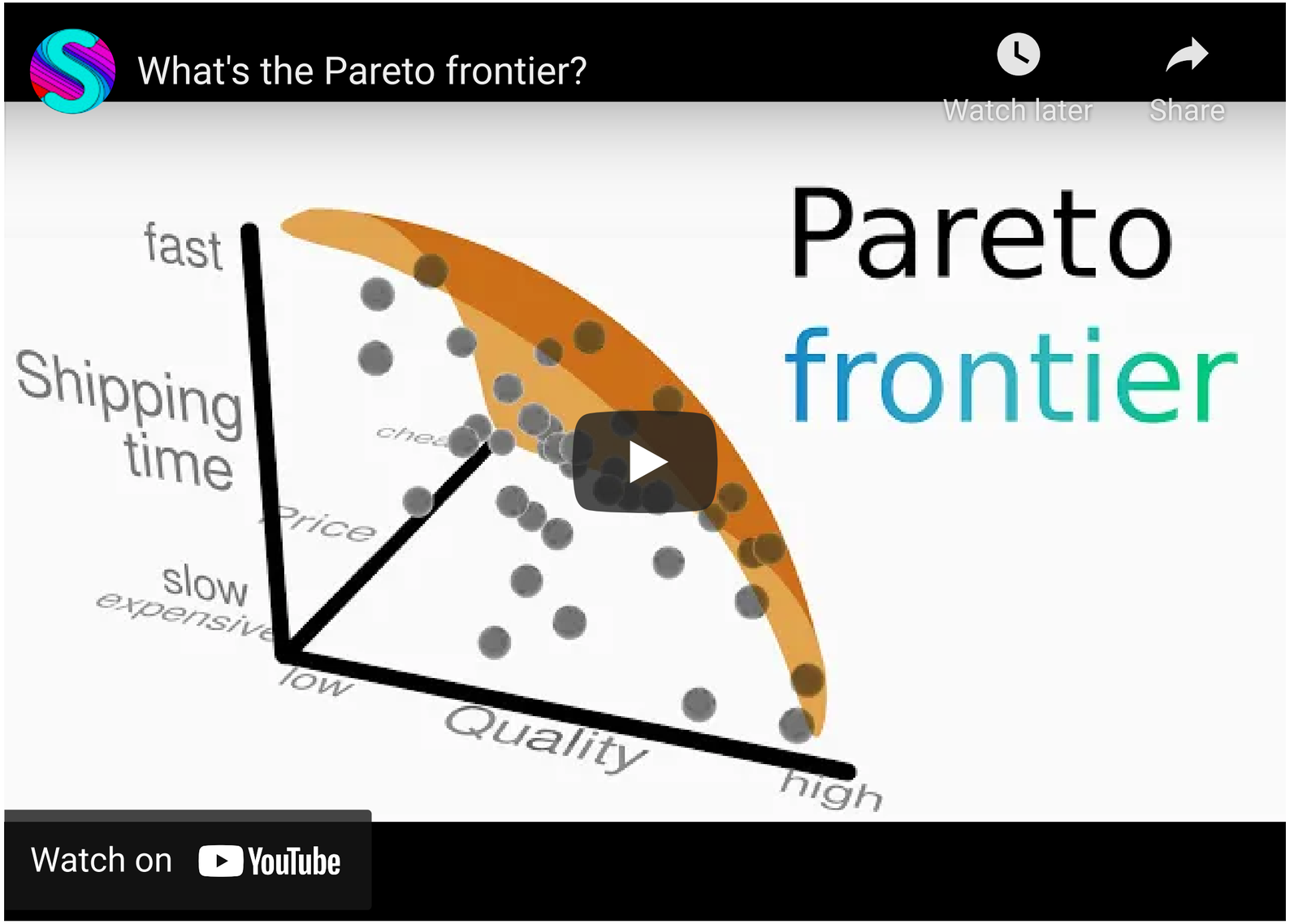
Rank	Model	Accuracy
1	**SMART-RoBERTa Large**	97.5
2	**T5-3B**	97.4
3	**MUPPET Roberta Large**	97.4
4	**ALBERT**	97.1
5	**T5-11B**	97.1
6	**StructBERTRoBERTa ensemble**	97.1

Pareto SOTA [1]



1. Towards Efficient NLP: A Standard Evaluation and A Strong Baseline [WIP]
   Fudan University, Huawei Poisson Lab

Pareto


What's the Pareto frontier?
Watch later
Share
Pareto
frontier
fast
Shipping time
slow
expensive
low
Quality
high
Watch on YouTube

ELUE

Efficient **L**anguage **U**nderstanding **E**valuation

on SST-2, IMDb, MRPC, STS-B, SNLI and SciTail

w.r.t.

1. performance (static & dynamic)
2. FLOPs [1] (static & dynamic)
3. number of parameters (static)

---

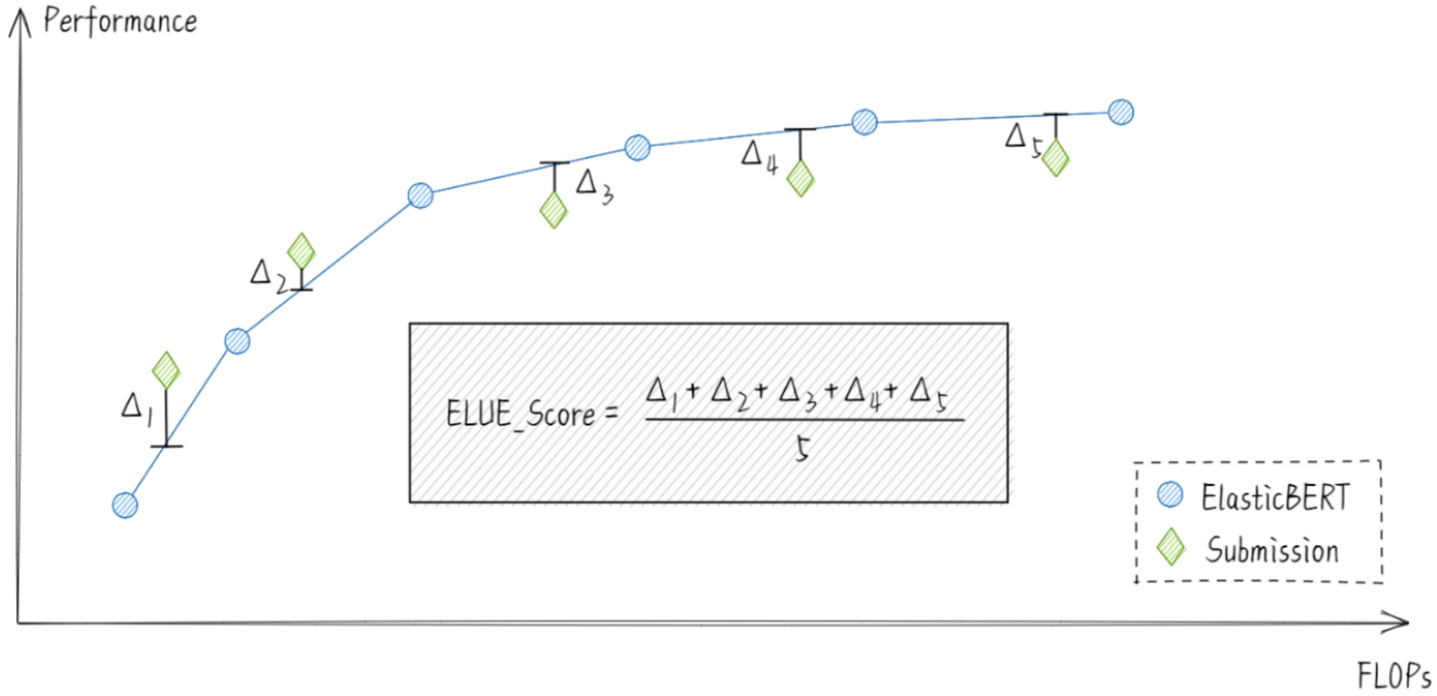
1. FLOPs and FLOPS are different!

   FLOPs: floating point operation**s**

   FLOPS: floating point operations **per second**

ELUE Score

12 layers of ElasticBERT → 12 discrete coordinates → continuous curve



$$\text{score} = \frac{1}{n}\sum_{i=1}^{n}\left[p_i - p^{EB}\left(f_i\right)\right]$$

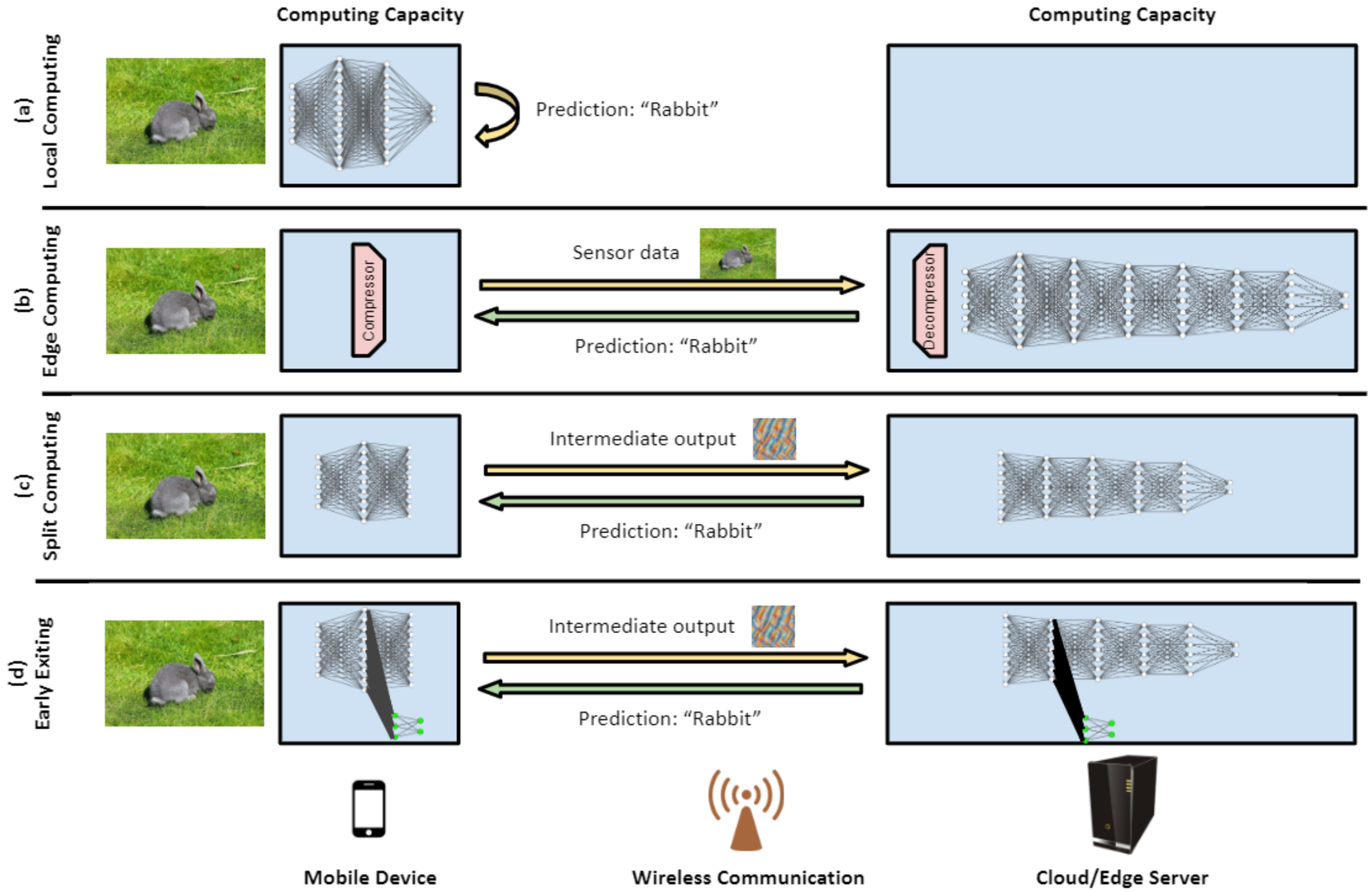
ELUE Leaderboard

	SST-2	IMDb	MRPC	STS-B	SNLI	SciTail	Average
ElasticBERT$_{\text{BASE}}$	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Static Models							
BERT$_{\text{BASE}}$	-4.6	-2.2	-5.9	-4.7	-1.5	-3.3	-3.7
ALBERT $_{\text{BASE}}$	-2.3	-1.1	**-0.3**	-2.6	-1.7	**-1.5**	**-1.6**
RoBERTa $_{\text{BASE}}$	**-0.9**	**-0.1**	-2.9	-5.1	**-0.7**	-3.3	-2.2
TinyBERT-6L	-1.7	-3.7	-2.6	**-1.6**	-0.8	-2.5	-2.2
Dynamic Models							
PABEE	-1.5	-0.3	-3.2	-2.3	-0.9	-0.6	-1.5
DeeBERT (BERT)	-12.3	-14.1	-5.2	-	-8.4	-6.4	-
DeeBERT (RoBERTa)	-2.3	-4.5	-3.1	-	-23.5	-9.9	-
ElasticBERT$_{\text{patience}}$	0.2	0.1	-1.1	**-0.3**	**0.0**	0.2	**-0.2**
ElasticBERT$_{\text{entropy}}$	**0.8**	**0.9**	**-0.4**	-	-0.1	**0.7**	-

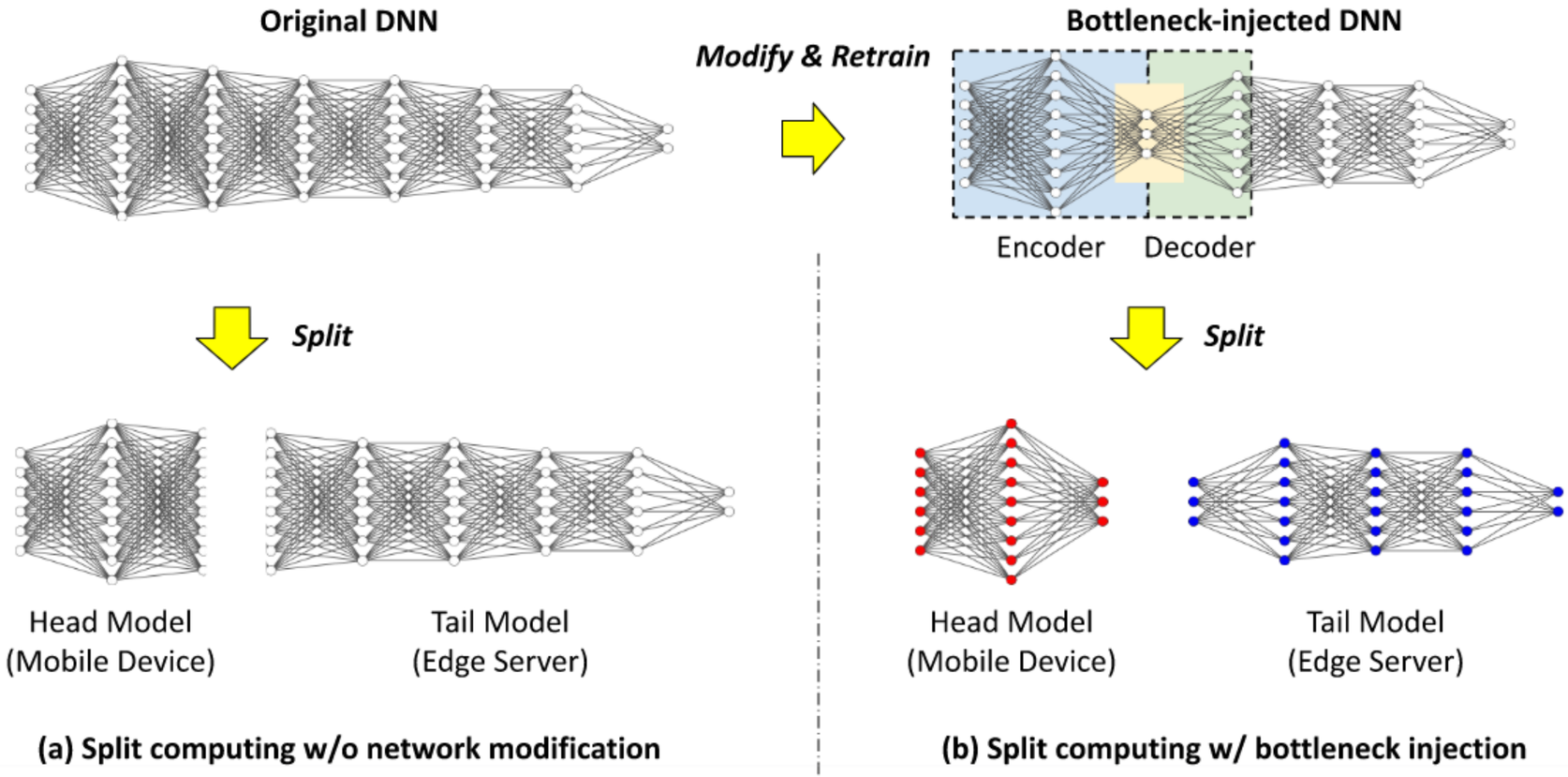
Recap & Insights

1. Complexity and confidence
    1. From single / multiple layers' logits
    2. From input features
    3. Learning-based
2. Training
    1. One-stage (joint) / two-stage (separate) fine-tuning
    2. Summed / averaged / weighted / re-scaled loss
    3. ~~Pre-training (ElasticBERT pre-trained on 64 V100)~~
3. Select backbone, evaluate with ELUE
4. Can EE really **dominate** their backbone? Why & how? (question from PABEE)
5. Combining compression a good idea?

Further — Split Computing



Further — Bottleneck Injection



1. Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges [WIP]