Paper 1: Distributed Inference on Mobile Edge and Cloud
A Data-Cartography based Clustering Approach[Link]

Divya Jyoti Bajpai and Manjesh Kumar Hanawal

Presentation by: Kishan Upadhyay [24n0453]
Guide: Prof Manjesh Kumar Hanawal

IEOR, IIT Bombay

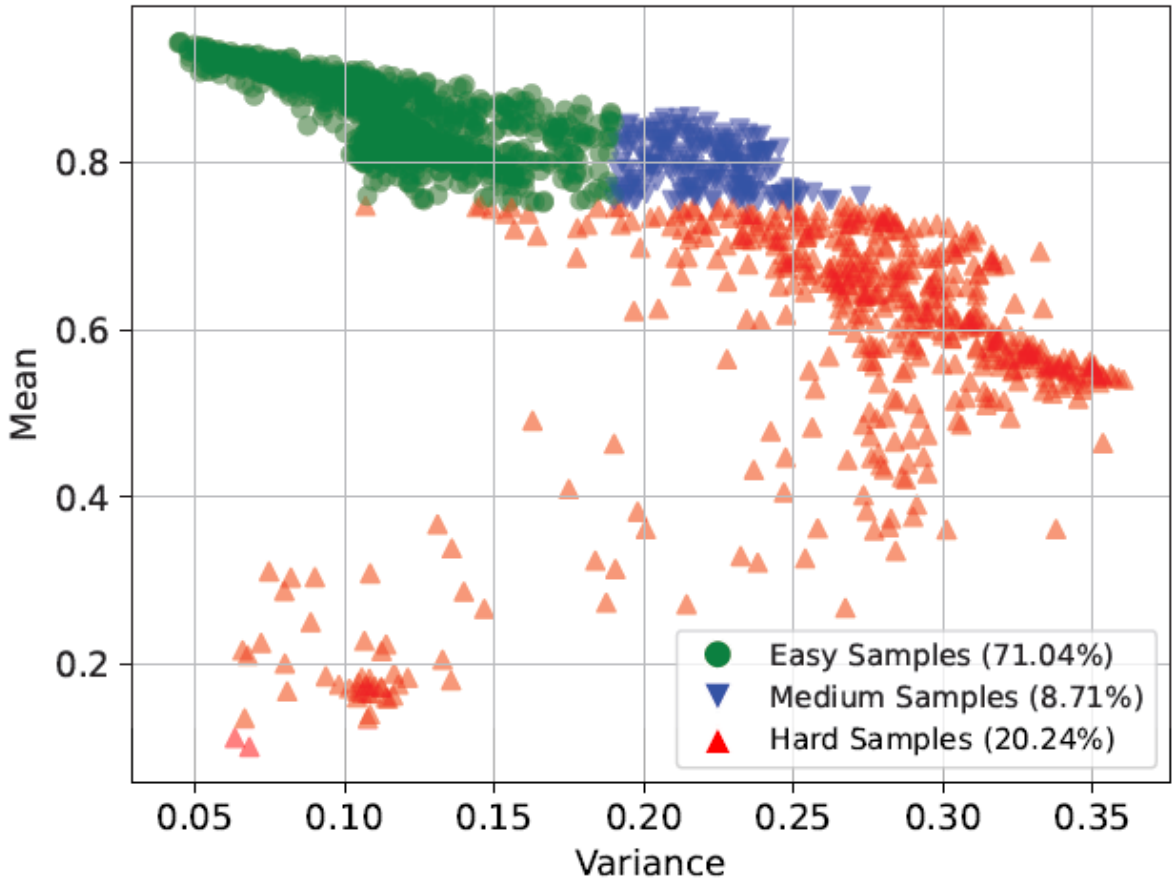
April 2025

Outline

1.1 Background and Motivation

- DNNs have shown outstanding performance across various domains
- **Challenge:** Large DNN models are difficult to deploy on resource-constrained devices
- **Current approaches:**
  - Model pruning, weight quantization, knowledge distillation
  - Early exits
  - Cloud offloading
- **Limitations:**
  - Compression techniques reduce accuracy
  - Cloud offloading increases latency and cost
- **Key insight:** Not all samples require the same computational effort!

1.2 Sample Complexity Varies



SST2 Dataset using BERT-base model

- **Easy samples:** High confidence, low variance (green)
- **Medium samples:** High confidence, high variance (blue)
- **Hard samples:** Low confidence (red)

1.3 Sample Classification

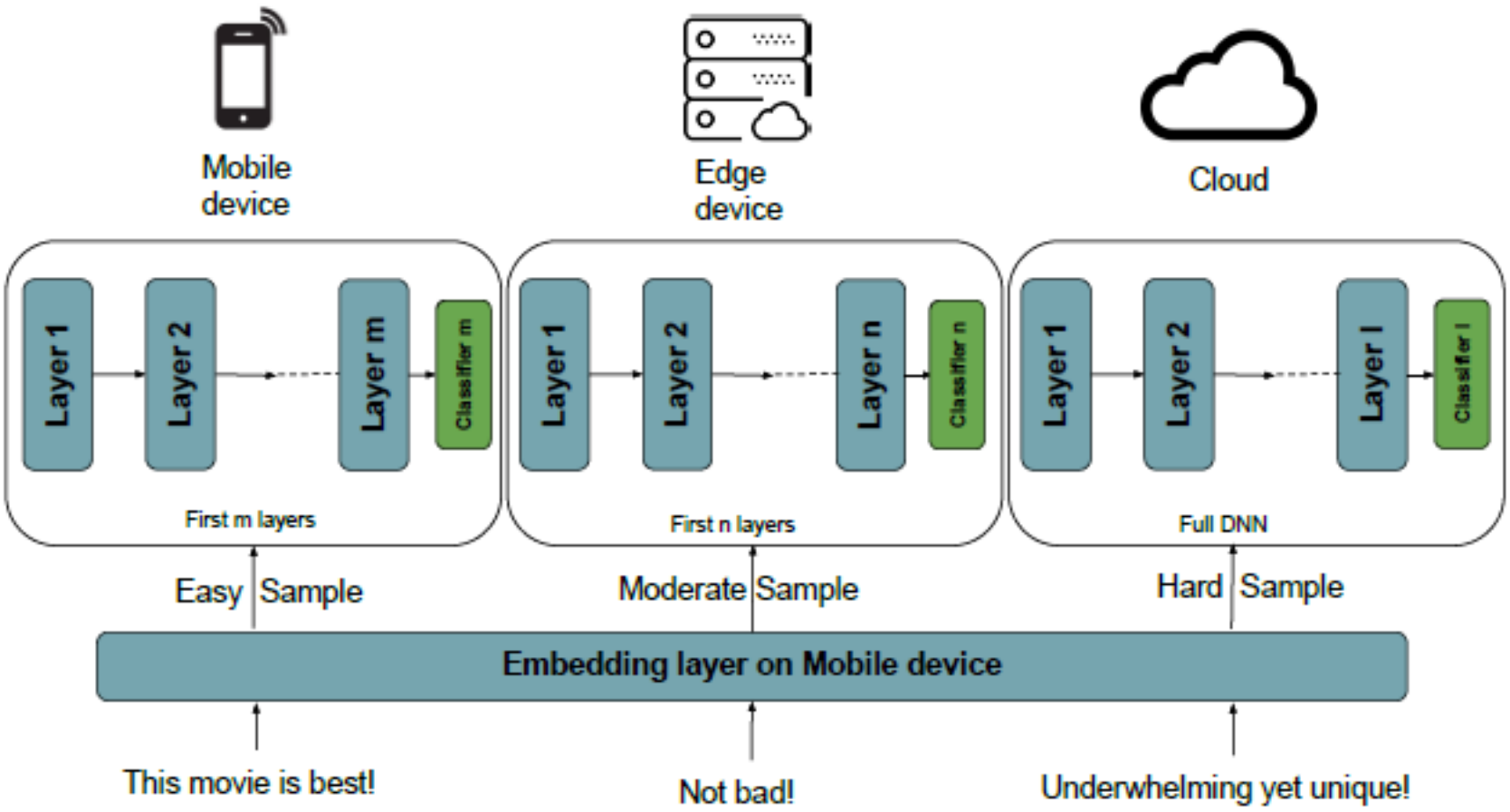
- **Easy samples:** High confidence, low variability
  - Model consistently predicts correctly
  - Can be processed on mobile device
- **Hard samples:** Low confidence, any variability
  - Model struggles with prediction
  - Need full model on cloud
- **Medium/Ambiguous samples:** High confidence, high variability
  - Model's confidence fluctuates
  - Suitable for edge processing

Example: In SST-2 dataset, 71.04% easy, 8.71% medium, 20.24% hard samples

1.4 Our Approach: DIMEC-DC

- **DIMEC-DC:** Distributed Inference on Mobile, Edge, and Cloud: A Data-Cartography based approach
- **Key Ideas:**
  - Deploy DNN sections progressively across devices:
    - Initial layers on mobile device
    - Larger portion on edge
    - Complete model on cloud
  - Use data cartography to assess sample complexity
  - Process samples on appropriate devices based on complexity
- **Benefits:**
  - Reduced costs ($> 43\%$)
  - Minimal accuracy drop ($< 0.5\%$)

1.5 System Architecture



Layer Distribution:

- Mobile device: First m layers
- Edge device: First n layers ($m < n < l$)
- Cloud: Complete model with l layers

2. Related Work

1. Split Computing

- Neurosurgeon
- BottleNet and Bottlefit
- BottleNet++
- CDE

Limitation: All samples offloaded regardless of complexity

2. Early Exit Methods

- AdaEE
- LEE, DEE
- UEE-UCB
- SplitEE, I-SplitEE

Limitation: All samples processed on mobile before deciding offloading

Paper's Contribution:

- First to use data cartography for distributed inference
- On-the-fly decision-making without processing on mobile first
- Selective offloading based on sample complexity

3. Data Cartography Overview

- Leverages training dynamics to categorize datasets
- Classifies samples based on confidence and variance

Key measures across training epochs:

- **Confidence:** Mean model probability of true label

$$\hat{\mu}_i = \frac{1}{E} \sum_{e=1}^{E} P(y_i^* | x_i, \theta_e) \tag{1}$$

- **Variability:** Variance of prediction probabilities

$$\hat{\sigma}_i = \frac{\sum_{e=1}^{E} (P(y_i^* | x_i, \theta_e) - \hat{\mu}_i)^2}{E} \tag{2}$$

- Taking a training dataset of size N, $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$

4.1 Training Exit Classifiers

- Start with pretrained BERT model, add classifiers at m^{th} (mobile) and n^{th} (edge) layer
- $(x, y^*) \sim \mathcal{D}$: input and h: classifier
- Train all classifiers simultaneously with cross-entropy loss

$$L_h(\theta) = L_{CE}(f_h(x, \theta), y^*) \tag{3}$$

- Overall loss: $L = L_m + L_e + L_c$
- E.g., For BERT-base:
  - Mobile: First 4 layers (TinyBERT)
  - Edge: First 6 layers (DistilBERT)
  - Cloud: Full 12 layers (BERT-base)

4.2 Pool Creation

Algorithm 1 Pool Creation

Input: x and thresholds α and β
Initialize $P_e = [], P_m = [], P_h = []$
for $e \leftarrow 1$ to E **do**
    Calculate $\mu_i = \hat{P}_m(y^*|x, \theta_e)$
end for
Calculate $\hat{\mu}_i, \hat{\sigma}_i$ using equations 1 and 2
if $\hat{\mu}_i \geq \alpha$ and $\hat{\sigma}_i < \beta$ **then**
    P_e.append(embed(x))
else if $\hat{\mu}_i \geq \alpha$ and $\hat{\sigma}_i \geq \beta$ **then**
    P_m.append(embed(x))
else
    P_h.append(embed(x))
end if
Return: P_e, P_m, P_h

4.3 Threshold Selection α, β

- Thresholds α and β determine pool assignment
- **Costs:**
  - Processing cost: λ_m (mobile), λ_e (edge)
  - Offloading cost: o_e (mobile to edge), o_c (mobile to cloud)
  - Cloud platform charge: γ
  - C_h: confidence score in the estimate at the h^{th} classifier
- **Reward function:**

$$r(\alpha,\beta) = \begin{cases} C_m - \lambda_m m & \text{if } \hat{\mu}_i \geq \alpha \text{ and } \hat{\sigma}_i \leq \beta \\ C_n - \lambda_e n - o_e & \text{if } \hat{\mu}_i \geq \alpha \text{ and } \hat{\sigma}_i > \beta \\ C_l - o_c - \gamma & \text{otherwise} \end{cases} \tag{4}$$

- Find (α,β) that maximizes expected reward on validation set, $\max_{(\alpha,\beta)\in S_1\times S_2} \mathbb{E}[r(\alpha,\beta)]$

$$\begin{aligned} \mathbb{E}[r(\alpha,\beta)] = {} & \mathbb{E}[C_m - \lambda_m i \mid \text{mobile inference}] \cdot P(\text{mobile inference}) \\ & + \mathbb{E}[C_n - \lambda_e - o_e \mid \text{edge inference}] \cdot P(\text{edge inference}) \\ & + \mathbb{E}[C_l - \gamma - o_c \mid \text{cloud inference}] \cdot P(\text{cloud inference}) \end{aligned}$$

4.4 Post-Deployment Inference

Fixed Inference:

- Create average embeddings for each pool: P_e^a, P_m^a, P_h^a
- For each new sample:
  - Compute embedding
  - Assign to nearest pool average
  - Process on corresponding device

Adaptive Inference:

- Same initial procedure as fixed
- Additionally, update pool averages with new samples:

$$P_e^a \leftarrow \frac{n_e \cdot P_e^a + x_e}{n_e + 1} \tag{5}$$

- Adapts to distribution shifts in test data

4.5 Dynamic Inference Algorithm

Algorithm 2 Dynamic Inference

Input: Test sample x, P_e^a, P_m^a and P_h^a
n_e, n_m, n_h = number of easy, medium and hard samples
$x_e \leftarrow \text{embed}(x)$
Calculate distance from all pool averages $d(x_e, \cdot)$
$d_{min}(x) \leftarrow \min\{d(x_e, P_e^a), d(x_e, P_m^a), d(x_e, P_h^a)\}$
if $d_{min} = d(x_e, P_e^a)$ **then**
    Infer the sample locally on mobile
    $P_e^a \leftarrow \frac{n_e \cdot P_e^a + x_e}{n_e + 1}$, $n_e \leftarrow n_e + 1$
else if $d_{min} = d(x_e, P_m^a)$ **then**
    Offload the sample to the edge and infer
    $P_m^a \leftarrow \frac{n_m \cdot P_m^a + x_e}{n_m + 1}$, $n_m \leftarrow n_m + 1$
else
    Offload the sample to the cloud and infer
    $P_h^a \leftarrow \frac{n_h \cdot P_h^a + x_e}{n_h + 1}$, $n_h \leftarrow n_h + 1$
end if

(A) Model Training (on the 80% Training Set)

- **Initialize** a pre-trained BERT (or similar) backbone.
- **Attach** two extra classifiers:
  - One after the m-th layer (mobile-level exit)
  - One after the n-th layer (edge-level exit)
- **Train** the entire network (all layers + both exits) for E epochs on the 80% training data.
- **Save** model weights at the end of each epoch:

$$\{\theta_1, \theta_2, \ldots, \theta_E\}$$

(B) Validation-Based Pool Creation (10% Validation Set)

Goal: quantify each sample's "complexity" (easy/medium/hard).

1. **For each epoch** $e = 1, \ldots, E$:

$$p_{i,e} \;=\; \widehat{P}_m\big(y_i^* \mid x_i, \theta_e\big)$$

   Run every validation sample x_i through the m-layer exit and record $p_{i,e}$.

2. **After all epochs**, for each sample i compute:

$$\widehat{\mu}_i \;=\; \frac{1}{E}\sum_{e=1}^{E} p_{i,e}, \quad \widehat{\sigma}_i^2 \;=\; \frac{1}{E}\sum_{e=1}^{E}\big(p_{i,e} - \widehat{\mu}_i\big)^2$$

3. **Grid-search** over thresholds α (confidence) and β (variance) to maximize the reward (Eq. 4).

4. **Assign** each validation embedding $\mathrm{embed}(x_i)$ into pools:

$$\begin{cases} \text{Easy: } \widehat{\mu}_i \geq \alpha \;\wedge\; \widehat{\sigma}_i < \beta, \\ \text{Medium: } \widehat{\mu}_i \geq \alpha \;\wedge\; \widehat{\sigma}_i \geq \beta, \\ \text{Hard: } \widehat{\mu}_i < \alpha. \end{cases}$$

(C) Inference on New Data (10% Test Set)

For each incoming test sample x:

1. Compute its word embedding $\mathrm{embed}(x)$ on the mobile device.
2. Calculate distances to each pool mean:

$$d_e = \left\|\mathrm{embed}(x) - \mathrm{mean}(P_e)\right\|, \quad d_m = \left\|\mathrm{embed}(x) - \mathrm{mean}(P_m)\right\|, \& \quad d_h$$

3. **Route** to the device with smallest distance:

$$\begin{cases} \text{Mobile exit (m-layer) if } d_e \text{ is smallest,} \\ \text{Edge exit (n-layer) if } d_m \text{ is smallest,} \\ \text{Cloud (full model) if } d_h \text{ is smallest.} \end{cases}$$

Adaptive mode : update that pool's mean with $\mathrm{embed}(x)$ to handle distribution shifts.

5.1 Baselines

We compare the model against following baselines:

- **BERT model:** Full model deployed on mobile (processing cost only)
- **Random:** Random assignment to devices
- **DeeBERT:** Early exit model deployed on cloud
- **AdaEE:** Adaptive early exit using multi-armed bandits
- **I-SplitEE:** Learns optimal splitting layer using multi-armed bandits
- **Ours-F:** Our method with fixed pool averages
- **Ours-D:** Our method with dynamic pool updates

5.2 Dataset Detailed

Dataset (Full Name)	**Description**
MRPC (Microsoft Research Paraphrase Corpus)	Determine whether two sentences are semantically equivalent.
QQP (Quora Question Pairs)	Identify if two questions from Quora are paraphrases of each other.
SST-2 (Stanford Sentiment Treebank)	Classify the sentiment of a sentence as either positive or negative.
CoLA (Corpus of Linguistic Acceptability)	Judge whether a sentence is grammatically acceptable in English.
QNLI (Question Natural Language Inference)	Determine whether a context sentence contains the answer to a given question.
MNLI (Multi-Genre Natural Language Inference)	Classify the relationship (entailment, neutral, contradiction) between a premise and a hypothesis across multiple genres.

Table: GLUE Benchmark Dataset

5.3 Experimental Setup

Datasets: GLUE benchmark

- **Semantic Equivalence:** MRPC, QQP
- **Sentiment Classification:** SST-2
- **Linguistic Acceptability:** CoLA
- **Natural Language Inference:** QNLI, MNLI

Implementation Details:

- Backbone models: BERT-base (12 layers), BERT-large (24 layers)
- Data split: 80% training, 10% validation, 10% test
- Training: 5 epochs, early stopping based on validation
- Hyperparameters: grid search over batch size {8, 16, 32}, learning rates {1e-5, 2e-5, 3e-5, 4e-5, 5e-5}

5.4 Cost Structure

Threshold search space $\alpha * \beta$:

- $\alpha \in \{0.55, 0.6, 0.65, 0.7, 0.8\}$
- $\beta \in \{0.05, 0.08, 0.11, 0.14, 0.17\}$

Layer distribution:

- **BERT-base:** $m = 4,\ n = 6,\ l = 12$
- **BERT-large:** $m = 6,\ n = 12,\ l = 24$

Cost values (relative to edge processing λ):

- Mobile processing: $\lambda_m = (3/2)\lambda$
- Edge processing: $\lambda_e = \lambda$
- Mobile-to-edge offloading: $o_e = (5/2)\lambda$
- Mobile-to-cloud offloading: $o_c = 3\lambda$

5.5 Main Results - BERT-base

Model	SST-2		CoLA		MNLI	
	Acc	Cost	Acc	Cost	Acc	Cost
BERT	93.5	1.00	58.3	1.00	84.5	1.00
Random	89.5	-27%	55.7	-31%	79.9	-46%
DeeBERT	91.0	-23%	56.5	-25%	82.1	-31%
AdaEE	92.1	-36%	56.9	-40%	82.8	-42%
I-SplitEE	92.4	-45%	57.3	-39%	83.6	-48%
Ours-F	93.0	-55%	57.3	-45%	83.9	-52%
Ours-D	**93.2**	**-61%**	**57.9**	**-50%**	**84.1**	**-59%**

Model	MRPC		QNLI		QQP	
	Acc	Cost	Acc	Cost	Acc	Cost
BERT	89.2	1.00	92.5	1.00	72.1	1.00
Random	86.5	-39%	89.6	-49%	69.4	-32%
DeeBERT	87.6	-42%	90.2	-36%	70.0	-28%
AdaEE	88.1	-51%	91.4	-41%	70.8	-30%
I-SplitEE	88.5	-58%	91.9	-57%	71.3	-39%
Ours-F	88.4	-65%	91.7	-61%	71.1	-36%
Ours-D	**88.7**	**-62%**	**92.1**	**-64%**	**72.2**	**-43%**

5.6 Main Results - BERT-large

Model	CoLA		MRPC		QNLI	
	Acc	Cost	Acc	Cost	Acc	Cost
BERT	59.5	1.00	90.1	1.00	93.1	1.00
Random	55.9	-45%	85.2	-59%	90.4	-51%
DeeBERT	56.2	-32%	88.2	-48%	91.2	-55%
AdaEE	58.1	-54%	89.2	-57%	91.7	-62%
I-SplitEE	58.4	-57%	88.9	-58%	92.4	-68%
Ours-F	58.5	-68%	89.7	-63%	92.1	-70%
Ours-D	**58.9**	**-71%**	**90.1**	**-58%**	**92.5**	**-75%**

- Our method consistently outperforms baselines
- Higher cost reduction for BERT-large ($> 70\%$ vs $> 60\%$ for BERT-base)
- Only 0.2-0.6% accuracy drop from full cloud inference
- QQP dataset: our method even outperforms vanilla BERT (overcomes "overthinking" issue)

6.1 Cost Variations

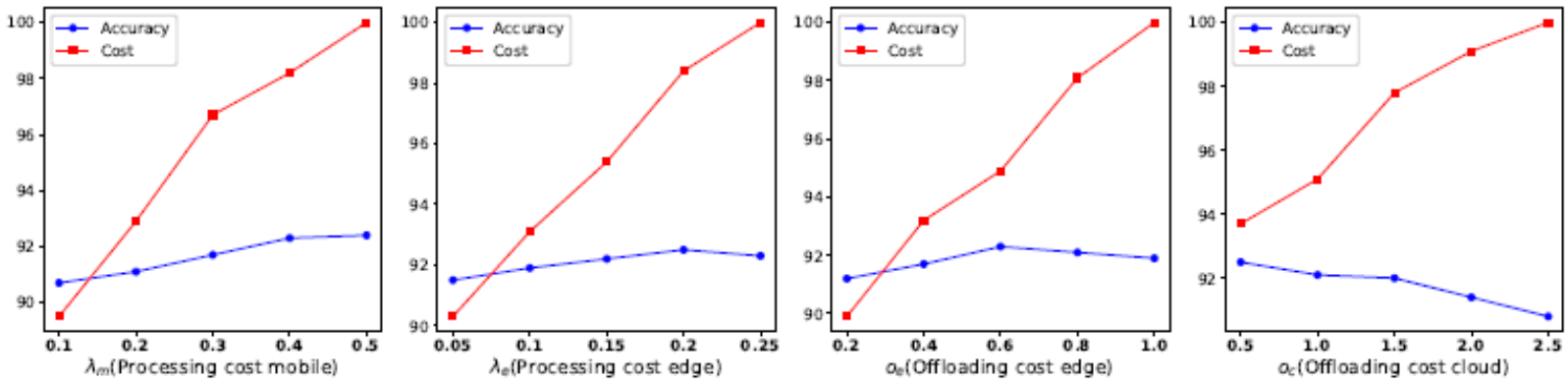
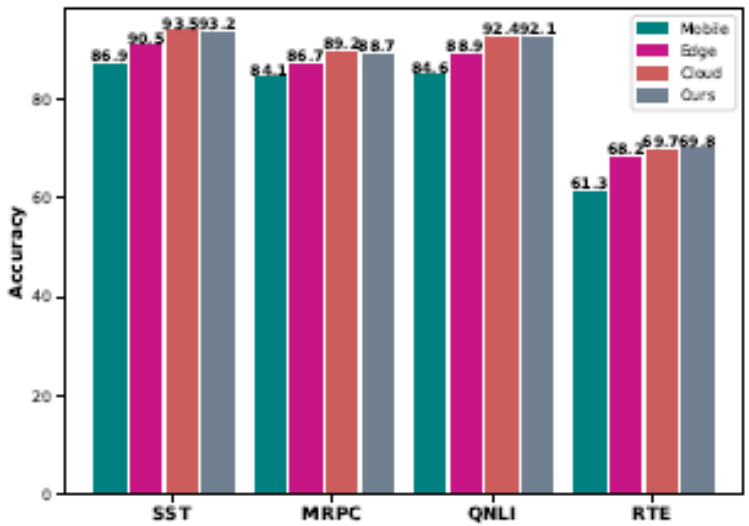


Fig. 5: The changes in accuracy and percentage change in cost values when one of the costs is varied while keeping others at a constant value.

Key findings:

- Model is robust to varying cost parameters
- Higher mobile processing cost (λ_m): slightly improved accuracy as more samples routed to edge/cloud
- Higher cloud offloading cost (o_c): small accuracy drop as more samples processed locally

6.2 Individual Device Inference

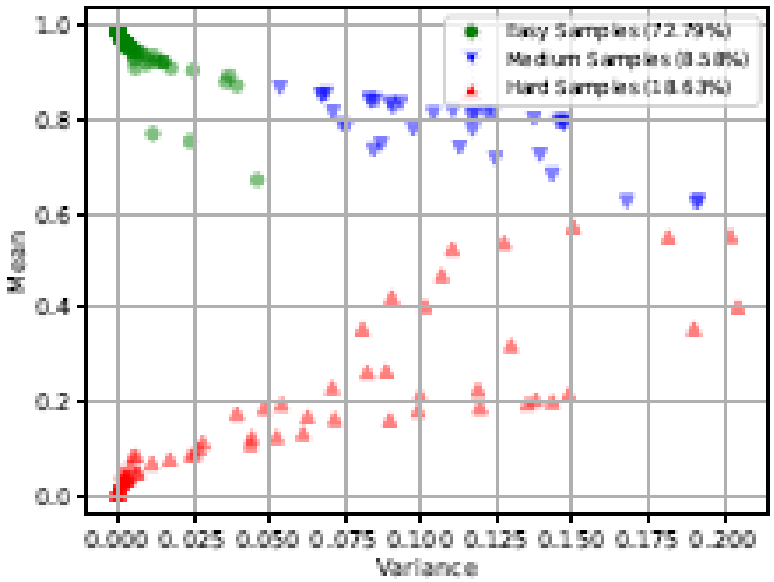


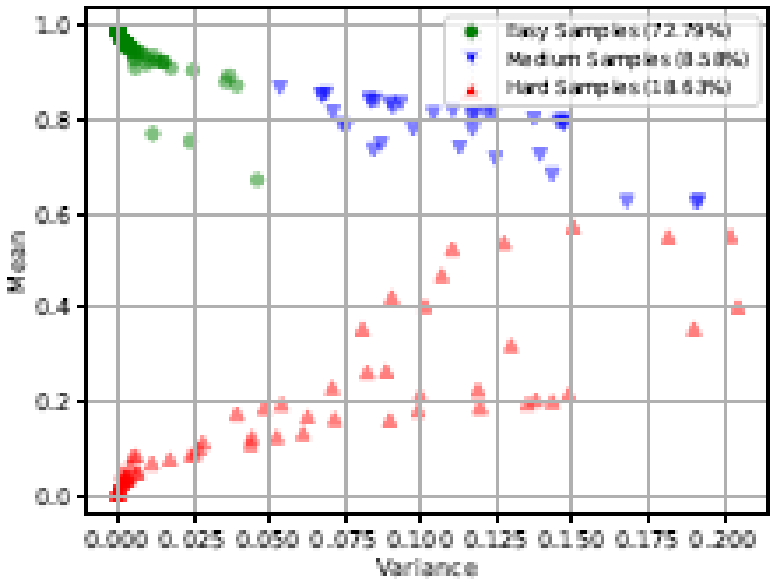
(a) Accuracy of Mobile, Edge and Cloud

Key observations:

- Mobile device alone: lowest accuracy
- Edge improves over mobile but still limited
- Cloud has highest accuracy (full model)
- Our method:
  - Nearly matches cloud accuracy
  - Sometimes exceeds cloud due to avoiding overthinking
  - Significantly reduces cost

6.3 Cluster Visualization

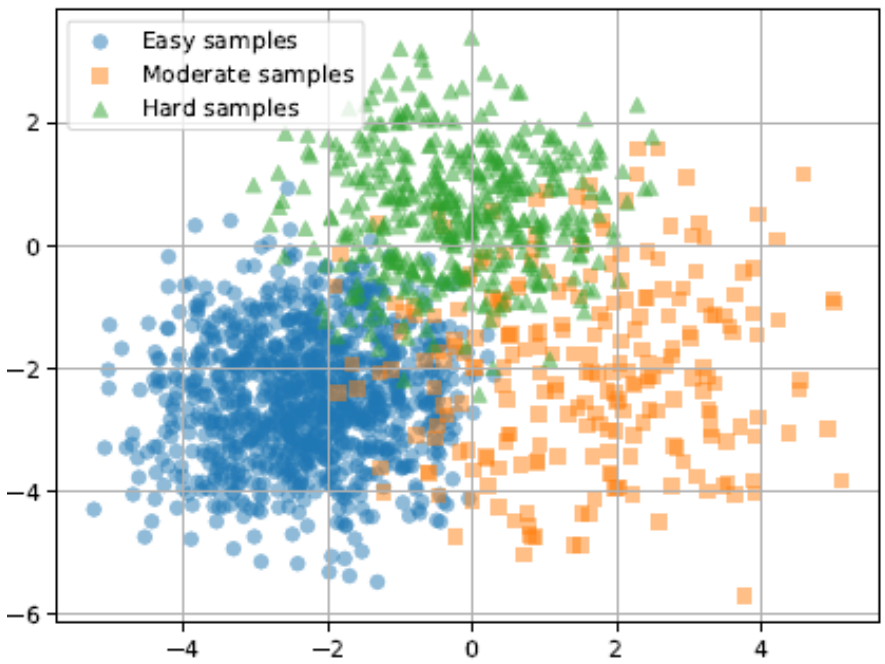




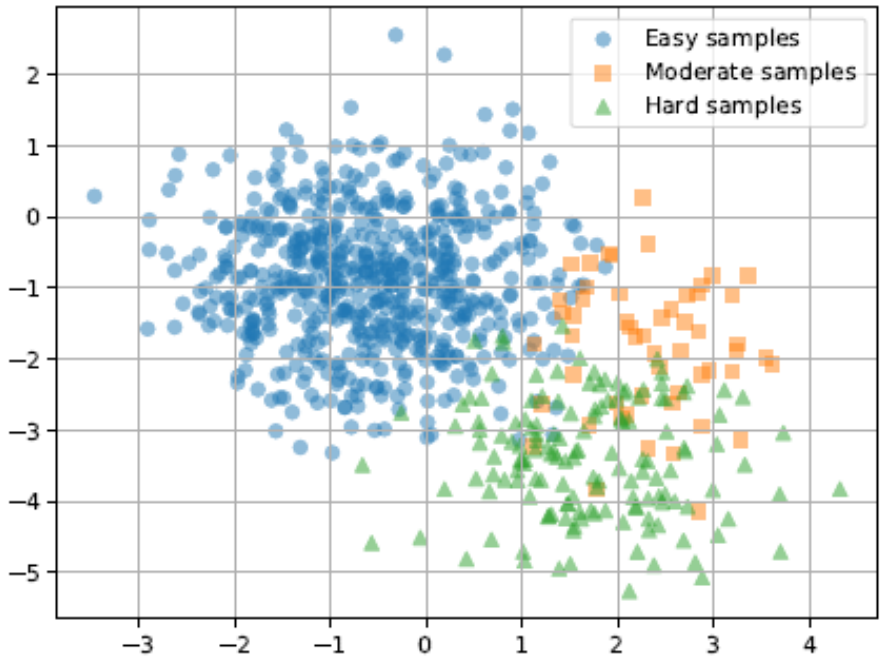
Dataset-specific clustering:

- Harder datasets (RTE, MRPC): more samples classified as "hard"
- Easier datasets (QNLI): more samples classified as "easy"
- Bell-shaped distribution across all datasets

6.4 t-SNE Visualization



QNLI



SST-2

Observations:

- Clear separation between easy and hard clusters
- Medium samples show higher overlap with other clusters
- Medium samples have higher spread (consistent with high variance definition)

Conclusion

Key Contributions:

- Leveraged data cartography for distributed inference
- Framework to assess sample complexity for appropriate resource allocation
- Significant cost reduction ($> 43\%$) with minimal accuracy drop ($< 0.5\%$)
- Robustness to changes in cost structure and device capabilities

Advantages:

- No on-device processing required before offloading decision
- No parameter reduction from backbone model
- Adaptive handling of distribution shifts

Paper 2: Distributed Inference on Mobile Edge and Cloud
An Early Exit based Clustering Approach[Link]

Divya Jyoti Bajpai and Manjesh Kumar Hanawal

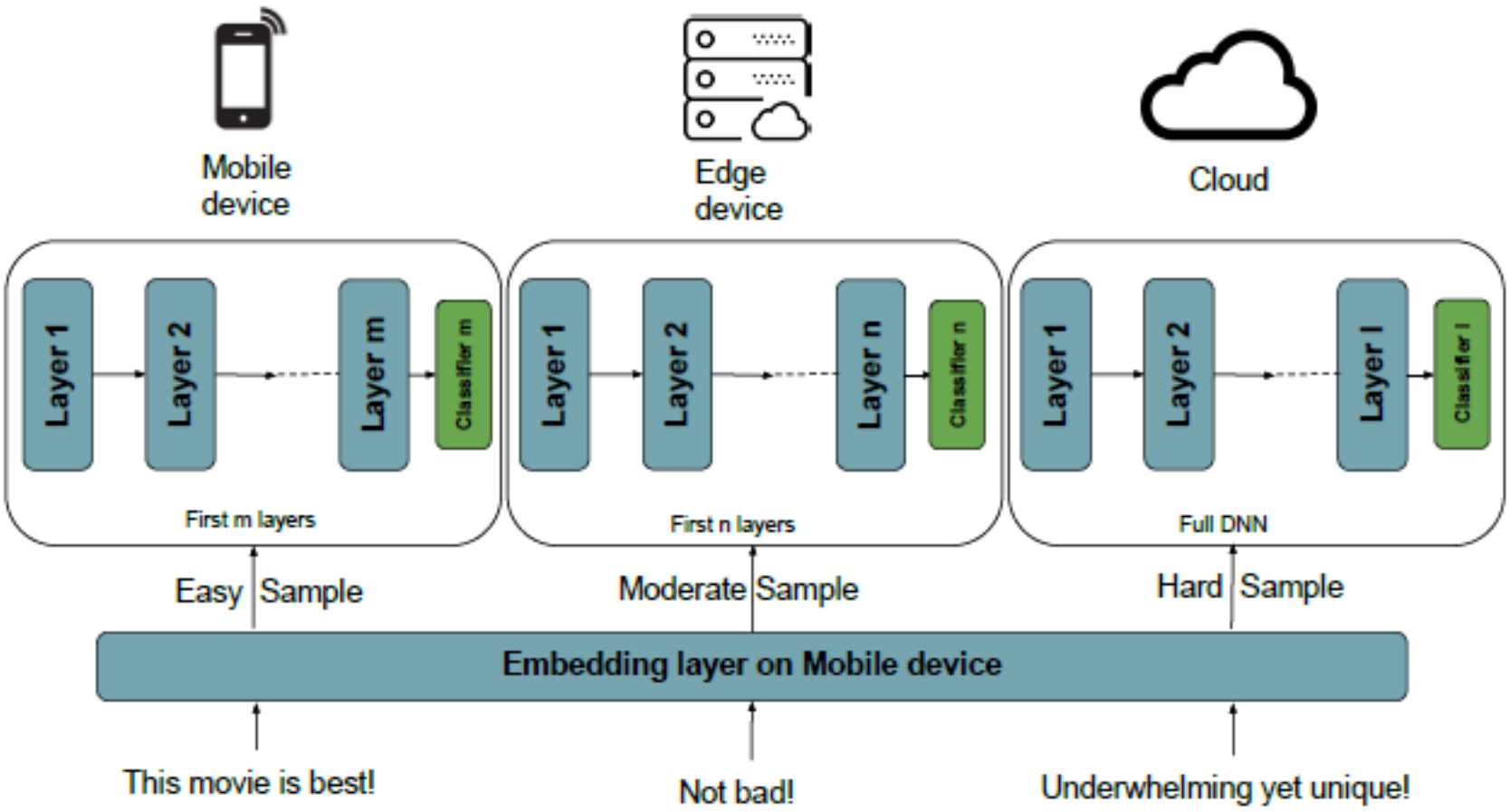
Presentation by: Kishan Upadhyay [24n0453]
Guide: Prof Manjesh Kumar Hanawal

IEOR, IIT Bombay

April 2025

Outline

1.1 System Architecture



Layer Distribution:

- Mobile device: First m layers
- Edge device: First n layers ($m < n < l$)
- Cloud: Complete model with l layers

1.2 Sample Classification

Training-time Pool Creation

- **Easy samples**: exit at any mobile exit before the mobile final layer (i.e., $\exists\, i \in \{1, \ldots, m\} : C_i \geq \alpha$).
- **Moderate samples**: exit at any edge exit before the edge final layer but not on mobile (i.e., $\forall\, i \leq m : C_i < \alpha$ and $\exists\, i \in \{m+1, \ldots, n\} : C_i \geq \alpha$).
- **Hard samples**: no early exit before the edge (i.e., $\forall\, i \leq n : C_i < \alpha$); processed by the full model on the cloud.

Inference-time Classification

- Compute the embedding x_e of each incoming sample on the mobile device.
- Assign to the pool by finding the nearest mean embedding $(\bar{P}^e, \bar{P}^m, \bar{P}^h)$.

1.3 Training Exit Classifiers

- Starting point: Pre-trained Language Model (BERT)
- Add exit classifiers after each layer of the backbone
- At the i^{th} exit, the loss is computed as:

$$L_i(\theta) = L_{CE}(f_i(x, \theta), y) \tag{1}$$

- Learn all classifiers simultaneously with overall loss:

$$L = \sum_{i=1}^{I} L_i \tag{2}$$

- $\hat{P}_i(c)$ = estimated probability for class c
- $C_i = \max_{c \in C} \hat{P}_i(c)$ = confidence at the i^{th} layer

1.4 Preparation of Dataset Pool

Algorithm 1 Pool creation

Input: x and threshold α
Initialize $P_e = [], P_m = [], P_h = []$
Process the sample till layer m
if $C_i \geq \alpha$ for any $i \in \{1, 2, \ldots, m\}$ **then**
    P_e.append(embed(x))
else if $C_m < \alpha$ and $C_i \geq \alpha$ for $i \in \{m+1, \ldots n\}$ **then**
    Pass the sample till n^{th} layer
    P_m.append(embed(x))
else
    Pass the sample till the final layer
    P_h.append(embed(x))
end if
Return: P_e, P_m, P_h

1.5 Choice of Threshold α

- Threshold α impacts accuracy-efficiency trade-off and cost
- **Costs considered:**
  - λ_m: Processing cost per layer on mobile device
  - λ_e: Processing cost per layer on edge
  - o_e: Offloading cost from mobile to edge
  - o_c: Offloading cost from mobile to cloud
  - γ: Cloud platform charge per sample
- **Reward function:**

$$r(\alpha) = \begin{cases} C_i - \lambda_m i & \text{if } C_i \geq \alpha \text{ and } i \leq m \\ C_i - \lambda_e i - o_e & \text{if } C_i \geq \alpha \text{ and } m < i \leq n \\ C_l - o_c - \gamma & \text{otherwise} \end{cases} \tag{3}$$

1.6 Expected Reward Function

- Expected reward:

$$E[r(\alpha)] = E[C_i - \lambda_m i | \text{mob. inference}]P[\text{mob. inference}] \quad (4)$$
$$+ E[C_i - \lambda_e - o_e | \text{edge inference}]P[\text{edge inference}] \quad (5)$$
$$+ E[C_i - \gamma - o_c | \text{cloud inference}]P[\text{cloud inference}] \quad (6)$$

- Optimization objective:

$$\max_{\alpha \in S} E[r(\alpha)] \quad (7)$$

- S = set of possible choices for α values
- Probabilities of inference location depend on the value of α

1.7 Post-Deployment Inference: Adaptive Method

Algorithm 2 Dynamic Inference

Input: Test sample x, P_e^a, P_m^a and P_h^a
n_e, n_m, n_h = number of samples in validation split
$x_e \leftarrow \text{embed}(x)$
Calculate distance from all pool averages $d(x_e, \cdot)$
$d_{min}(x) \leftarrow \min\{d(x_e, P_e^a), d(x_e, P_m^a), d(x_e, P_h^a)\}$
if $d_{min} = d(x_e, P_e^a)$ **then**
    Infer the sample locally on mobile
    $P_e^a \leftarrow \frac{n_e \cdot P_e^a + x_e}{n_e + 1}$, $n_e \leftarrow n_e + 1$
else if $d_{min} = d(x_e, P_m^a)$ **then**
    Offload the sample to the edge and process
    $P_m^a \leftarrow \frac{n_m \cdot P_m^a + x_e}{n_m + 1}$, $n_m \leftarrow n_m + 1$
else
    Offload the sample to the cloud
    $P_h^a \leftarrow \frac{n_h \cdot P_h^a + x_e}{n_h + 1}$, $n_h \leftarrow n_h + 1$
end if

2.1 Datasets: GLUE

- Evaluation on GLUE datasets covering three types of tasks:
- **Sentiment Classification:**
  - SST-2: Stanford-Sentiment Treebank
  - CoLA: Corpus of Linguistic Acceptability
- **Entailment Classification:**
  - MRPC: Microsoft Research Paraphrase Corpus
  - QQP: Quora Question Pairs
- **Natural Language Inference:**
  - QNLI: Question-answering Natural Language Inference
  - MNLI: Multi-Genre Natural Language Inference

2.2 Baselines

- **BERT model:** Original BERT backbone deployed on mobile (processing cost only)
- **Random:** Random assignment of incoming samples to mobile/edge/cloud
- **Early-Exit:** Complete model on mobile with early exits
- **AdaEE:** Adaptive method using MABs to learn optimal threshold for offloading
- **I-SplitEE:** Learns optimal splitting layer based on accuracy-cost trade-off
- **Ours-F:** Fixed pool average method (no updates during inference)
- **Ours-D:** Dynamic pool average updates (adaptive method)

2.3 Training Procedure

- **Backbone model:** BERT-base/large with linear output layer after each intermediate layer
- **Dataset split:** 80% training, 10% validation, 10% test
- **Training details:**
  - 5 epochs
  - Grid search over batch sizes {8, 16, 32}
  - Learning rates {1e-5, 2e-5, 3e-5, 4e-5, 5e-5}
  - Adam optimizer
  - Early stopping based on validation performance

2.4 Pool Creation and Cost Structure

- Pool creation using validation split
- Values of m and n chosen based on cost structure:
  - $m = 3, 6$ and $n = 6, 12$ for BERT-base and large
- Threshold set S: ten equidistant values from $1/C$ to 1.0
  - C = number of classes
  - Values below $1/C$ are extraneous (definition of confidence)
- Cost structure (in terms of smallest unit λ):
  - $\lambda_2 = \lambda$ (edge processing cost)
  - $\lambda_1 = (3/2)\lambda$ (mobile processing cost)
  - $o_1 = (5/2)\lambda$ (mobile to edge offloading)
  - $o_2 = 3\lambda$ (mobile to cloud offloading)

2.5 Inference Process

- Batch size of 1 (sequential data arrival)
- Process for each incoming sample:
  1. Calculate word embedding on mobile device
  2. Compute distance against pool averages
  3. Assign to closest pool (easy, moderate, or hard)
  4. Based on assignment:
     - Easy $\rightarrow$ infer on mobile device
     - Moderate $\rightarrow$ offload to edge device
     - Hard $\rightarrow$ offload to cloud
- Hardware: NVIDIA RTX 2070 GPU
- Runtime: Average 3 hours, maximum 10 hours (MNLI dataset)

3.1 Main Results: BERT-base

Model/Data	SST-2		CoLA		MNLI		MRPC		QNLI		QQP	
	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost
BERT	93.5	1.00	58.3	1.00	84.5	1.00	89.2	1.00	92.5	1.00	72.1	1.00
Random	89.5	-27%	55.7	-31%	79.9	-46%	86.5	-39%	89.6	-49%	69.4	-32%
Early-Exit	91.0	-23%	56.5	-25%	82.1	-31%	87.6	-42%	90.2	-36%	70.0	-28%
AdaEE	92.1	-36%	56.9	-40%	82.8	-42%	88.1	-51%	91.4	-41%	70.8	-30%
I-SplitEE	92.4	-45%	57.3	-39%	83.6	-48%	88.5	-58%	91.9	-57%	71.3	-39%
Ours-F	93.3	-43%	57.8	-42%	83.9	-53%	88.9	-62%	82.1	-58%	71.8	-44%
Ours-D	**93.6**	**-47%**	**58.1**	**-43%**	**84.3**	**-57%**	**89.2**	**-61%**	**92.2**	**-63%**	**72.0**	–47%

Table: Results on BERT-base backbone

- DIMEE outperforms all baselines in both accuracy and cost
- Ours-D achieves significant cost reduction (¿47%) with minimal accuracy drop (sometimes even improved)
- Dynamic pool updating improves performance compared to fixed variant

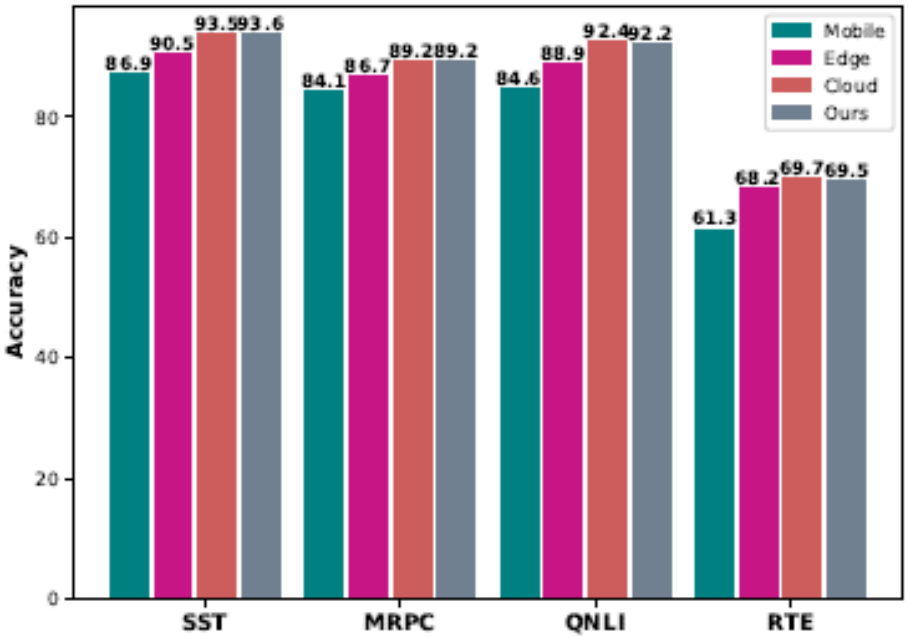
3.2 Main Results: BERT-large

Model/Data	CoLA		MRPC		QNLI	
	Acc	Cost	Acc	Cost	Acc	Cost
BERT	59.5	1.00	90.1	1.00	93.1	1.00
Random	55.9	-45%	85.2	-59%	90.4	-51%
Early-Exit	56.2	-32%	88.2	-48%	91.2	-55%
AdaEE	58.1	-54%	89.2	-57%	91.7	-62%
I-SplitEE	58.4	-57%	88.9	-58%	92.4	-68%
Ours-F	58.9	-61%	89.7	-63%	92.6	-72%
Ours-D	**59.2**	**-65%**	**90.2**	**-67%**	**92.8**	**-75%**

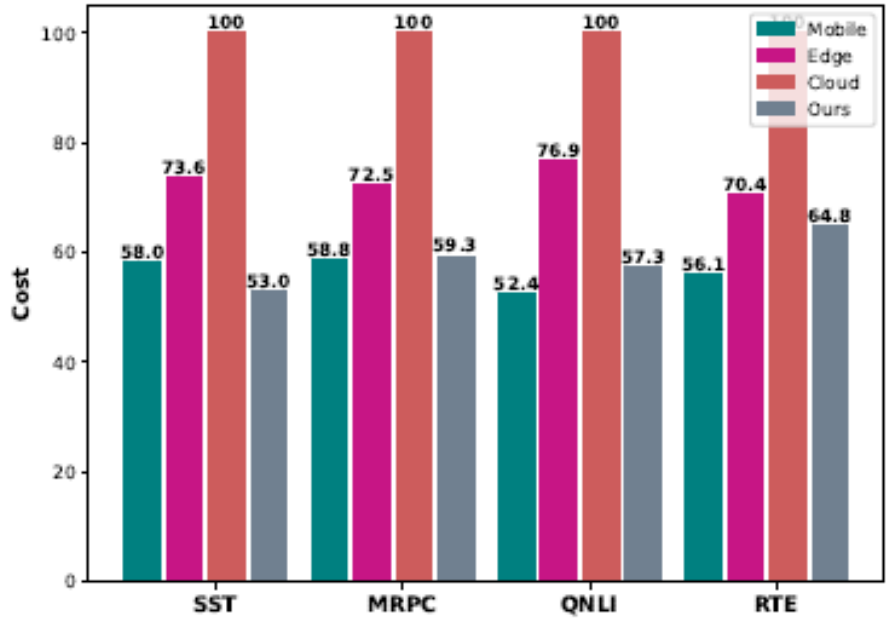
Table: Results on BERT-large variant

- Cost reduction even larger for BERT-large (overparameterized)
- Similar pattern of performance across different models
- Adaptive method (Ours-D) consistently outperforms fixed approach

4.1 Individual Device Inference



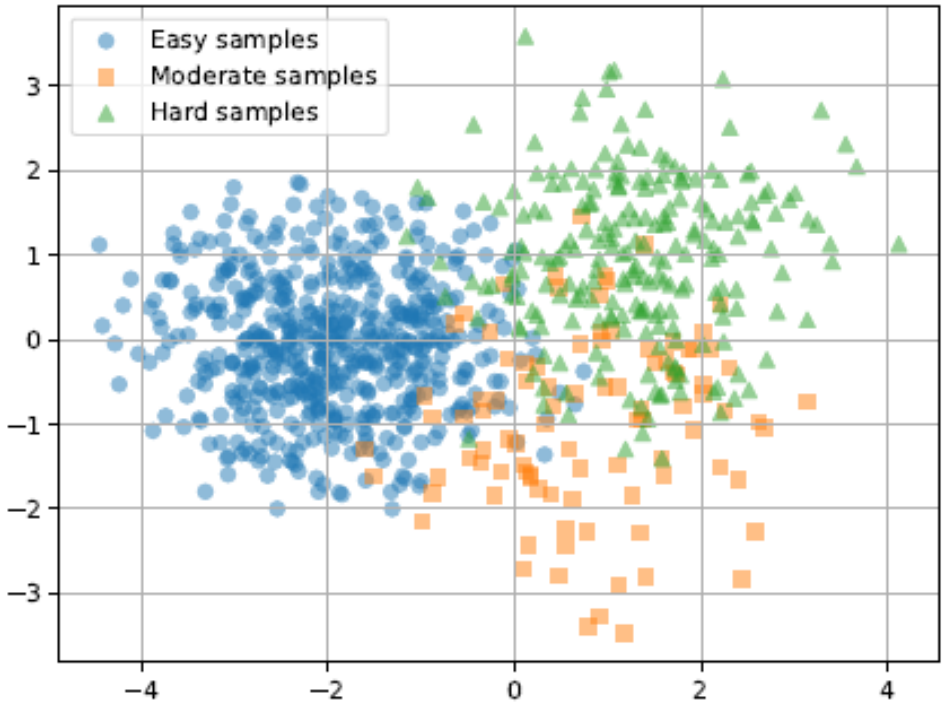
(a) Accuracy of Mobile, Edge and Cloud



(b) Cost of Mobile, Edge and Cloud

- Cloud has highest accuracy (full-fledged DNN) but also highest cost
- Our method sometimes outperforms full cloud DNN due to avoiding overthinking

4.2 t-SNE Visualization of Embeddings



(c) The t-SNE visualization of embeddings.

- Clear separation between easy, moderate, and hard sample embeddings
- Shows effectiveness of using embeddings for complexity determination
- Validates the clustering approach for sample assignment

4.3 Cost Variations

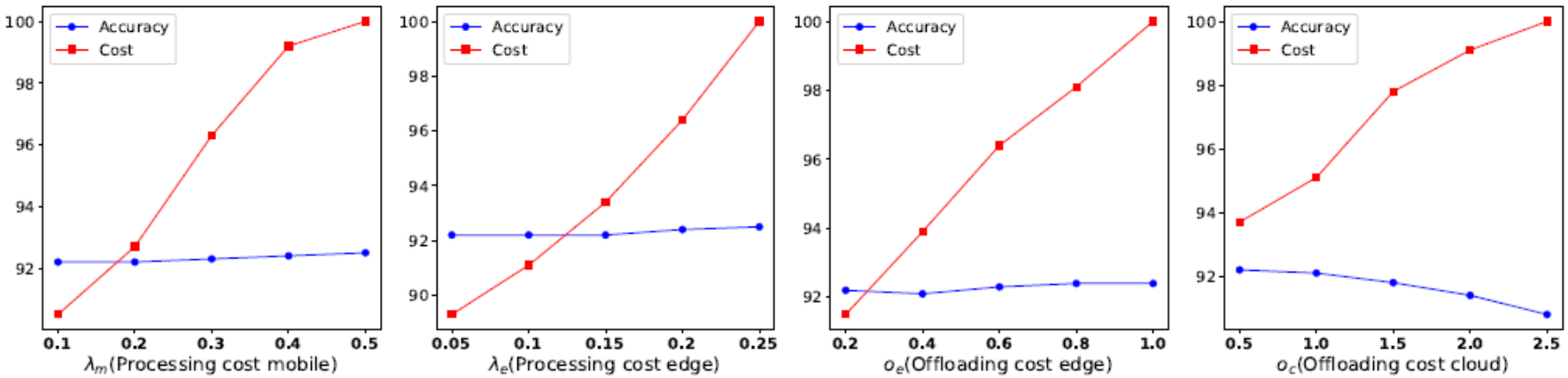


Fig. 3: The changes in accuracy and percentage change in cost values when one of the cost is varied while keeping others at a constant value.

- **Mobile processing cost (λ_m):** Increasing forces more offloading to edge/cloud, slightly improving accuracy
- **Edge processing cost (λ_e):** Similar effect as increasing mobile cost
- **Cloud offloading cost (o_c):** Higher values discourage cloud offloading, slightly reducing accuracy
- Method is robust to cost variations (minimal accuracy change across different cost structures)

5. Conclusion

- Experimental results across NLP tasks show:
  - $>$ 43% cost reduction
  - $<$ 0.3% accuracy drop
- Optimizes accuracy-cost trade-off through intelligent sample assignment:
  - Easy tasks $\rightarrow$ more local inference
  - Hard tasks $\rightarrow$ more offloading to preserve accuracy
- Robust to various cost structures