Rethinking the Adversarial Robustness of Multi-Exit Neural Networks in an Attack-Defense Game

Supplementary Material

A. Nash Equilibrium of the Multi-exit Attack-Defense Game

A.1. A Proof of MiniMax Theorem

We provide a brief proof of the Minimax Theorem. With this theorem, we ensure that NEED can always find the best strategy for the defender and make a robust defense scheme.

Lemma A.1. *Let $C \subset \mathbb{R}^{|\mathcal{A}|}$ be a compact, convex set of an Euclidean space and $\mathbf{0} \notin C$, then $\exists \boldsymbol{z} \in \mathbb{R}^{|\mathcal{A}|}$ s.t.*

$$\boldsymbol{z}^\top \boldsymbol{x} \geq 0 \quad \forall \boldsymbol{x} \in C$$

Proof. Let the point $\boldsymbol{z}' \in C$ that is nearest to the origin in C, then $\forall \boldsymbol{x} \in C$, we have $\|\boldsymbol{z}'\|_2^2 \leq \|\boldsymbol{x}\|_2^2$. By definition of convexity, we have $(1-\alpha)\cdot\boldsymbol{z}'+\alpha\cdot\boldsymbol{x} \in C$, where $\alpha \in (0,1)$, then

$$\begin{aligned}
\|\boldsymbol{z}'\|_2^2 &\leq \|(1-\alpha)\cdot\boldsymbol{z}' + \alpha\cdot\boldsymbol{x}\|_2^2 \\
\|\boldsymbol{z}'\|_2^2 &\leq (1-\alpha)^2\|\boldsymbol{z}'\|_2^2 + \alpha^2\|\boldsymbol{x}\|_2^2 + 2\alpha(1-\alpha)(\boldsymbol{z}')^\top\boldsymbol{x} \\
0 &\leq \alpha(\alpha-2)\|\boldsymbol{z}'\|_2^2 + \alpha^2\|\boldsymbol{x}\|_2^2 + 2\alpha(1-\alpha)(\boldsymbol{z}')^\top\boldsymbol{x} \\
0 &\leq (\alpha-2)\|\boldsymbol{z}'\|_2^2 + \alpha\|\boldsymbol{x}\|_2^2 + 2(1-\alpha)(\boldsymbol{z}')^\top\boldsymbol{x}
\end{aligned}$$

let $\alpha \to 0$, then

$$\begin{aligned}
-2\|\boldsymbol{z}'\|_2^2 + 2(\boldsymbol{z}')^\top\boldsymbol{x} &\geq 0 \\
(\boldsymbol{z}')^\top\boldsymbol{x} &\geq \|\boldsymbol{z}'\|_2^2 \\
(\boldsymbol{z}')^\top\boldsymbol{x} &\geq 0
\end{aligned}$$

Then we have $\boldsymbol{z}'$ satisfying $\boldsymbol{z}$ in Lemma A.1. □

Lemma A.2. *Let $\mathbf{M}$ be any matrix of order $|\mathcal{A}| \times |\mathcal{A}|$, then either*

(1) $\exists \boldsymbol{x} \in \mathbb{R}^{|\mathcal{A}|}$, $\boldsymbol{x} \neq \mathbf{0}$ and $\boldsymbol{x} \succeq \mathbf{0}$, *s.t.*

$$\boldsymbol{x}^\top\mathbf{M} \succeq \mathbf{0},$$

(2) $\exists \boldsymbol{y} \in \mathbb{R}^{|\mathcal{A}|}$, $\boldsymbol{y} \neq \mathbf{0}$ and $\boldsymbol{y} \succeq \mathbf{0}$, *s.t.*

$$\mathbf{M}\boldsymbol{y} \preceq \mathbf{0}$$

Proof. Let $\boldsymbol{e}_1, \boldsymbol{e}_2, \cdots, \boldsymbol{e}_{|\mathcal{A}|}$ be the standard unit vectors in $\mathbb{R}^{|\mathcal{A}|}$, where $\boldsymbol{e}_i = [0, \cdots, 0, 1, 0, \cdots, 0]$ with only the i-th element being 1. Let the rows of $\mathbf{M}$ be denoted by $\boldsymbol{m}_1, \boldsymbol{m}_2, \cdots, \boldsymbol{m}_{|\mathcal{A}|} \in \mathbb{R}^{|\mathcal{A}|}$.

Let C be the convex hull of $\{-\boldsymbol{e}_1, -\boldsymbol{e}_2, \cdots, -\boldsymbol{e}_{|\mathcal{A}|}, \boldsymbol{m}_1, \boldsymbol{m}_2, \cdots, \boldsymbol{m}_{|\mathcal{A}|}\}$, then there are two cases: $\mathbf{0} \in C$ or $\mathbf{0} \notin C$.

Case I. $\mathbf{0} \in C$. $\exists \alpha_1, \alpha_2, \cdots, \alpha_{|\mathcal{A}|}, \beta_1, \beta_2, \cdots, \beta_{|\mathcal{A}|} \in \mathbb{R}$ that are non-negative, *s.t.*

$$\begin{cases}
-\beta_1\boldsymbol{e}_1 - \beta_2\boldsymbol{e}_2 - \cdots - \beta_{|\mathcal{A}|}\boldsymbol{e}_{|\mathcal{A}|} \\
+\alpha_1\boldsymbol{m}_1 + \alpha_2\boldsymbol{m}_2 + \cdots + \alpha_{|\mathcal{A}|}\boldsymbol{m}_{|\mathcal{A}|} = \mathbf{0} \\
\beta_1 + \beta_2 + \cdots + \beta_{|\mathcal{A}|} + \alpha_1 + \alpha_2 + \cdots + \alpha_{|\mathcal{A}|} = 1
\end{cases}$$

Now we show that not all $\alpha_i, i = 1, 2, \cdots, |\mathcal{A}|$ are 0. Suppose that $\forall \alpha_i, i = 1, 2, \cdots, |\mathcal{A}|, \alpha_i = 0$, then

$$\begin{cases}
\beta_1\boldsymbol{e}_1 + \beta_2\boldsymbol{e}_2 + \cdots + \beta_{|\mathcal{A}|}\boldsymbol{e}_{|\mathcal{A}|} = \mathbf{0} \\
\beta_1 + \beta_2 + \cdots + \beta_{|\mathcal{A}|} = 1
\end{cases}$$

which are contradictory since $\boldsymbol{e}_1, \boldsymbol{e}_2, \cdots, \boldsymbol{e}_{|\mathcal{A}|}$ are standard unit vectors, so not all $\alpha_i, i = 1, 2, \cdots, |\mathcal{A}|$ are 0. Therefore, we have none negative $\alpha_1, \alpha_2, \cdots, \alpha_{|\mathcal{A}|}$ (not all-zero) $\in \mathbb{R}$ *s.t.*

$$\alpha_1\boldsymbol{m}_1 + \alpha_2\boldsymbol{m}_2 + \cdots + \alpha_{|\mathcal{A}|}\boldsymbol{m}_{|\mathcal{A}|} = \boldsymbol{z}$$

where $\boldsymbol{z} = [\beta_1, \beta_2, \cdots, \beta_{|\mathcal{A}|}]^\top \in \mathbb{R}^{|\mathcal{A}|}$ and $\boldsymbol{z} \succeq \mathbf{0}$. Then $\boldsymbol{x} = [\alpha_1, \alpha_2, \cdots, \alpha_{|\mathcal{A}|}]^\top$ satisfies $\boldsymbol{x}^\top\mathbf{M} \succeq \mathbf{0}$, which proves the first alternative in Lemma A.2.

Case II. $\mathbf{0} \notin C$. By Lemma A.1, $\exists \boldsymbol{z}$ *s.t.* $\boldsymbol{z}^\top\boldsymbol{x} \geq 0, \forall \boldsymbol{x} \in C$. By definition of C, $-\boldsymbol{e}_i \in C$, then we have $\boldsymbol{z}^\top(-\boldsymbol{e}_i) \geq 0$, which implies that $z_i \leq 0$; also $\boldsymbol{m}_i \in C$, then we have $\boldsymbol{z}^\top\boldsymbol{m}_i \geq 0$, which implies

$$\mathbf{M}\boldsymbol{z} \succeq \mathbf{0} \text{ and } \boldsymbol{z} \preceq \mathbf{0}$$

Let $\boldsymbol{y} = -\boldsymbol{z}$, and we have

$$\mathbf{M}\boldsymbol{y} \preceq \mathbf{0} \text{ and } \boldsymbol{y} \succeq \mathbf{0}$$

that proves the second alternative of Lemma A.2. □

Theorem A.3 (Minimax Theorem)**.** *In the two-player zero-sum attack-defense game G with a strategy space $\mathcal{S}$, $\exists(\boldsymbol{s}^{a*}, \boldsymbol{s}^{d*}) \in \mathcal{S} \times \mathcal{S}$ such that $u^d(\boldsymbol{s}^{a*}, \boldsymbol{s}^d) \leq u^d(\boldsymbol{s}^{a*}, \boldsymbol{s}^{d*}) \leq u^d(\boldsymbol{s}^a, \boldsymbol{s}^{d*})$ for all $\boldsymbol{s}^a, \boldsymbol{s}^d \in \mathcal{S}$, i.e., achieves the Nash equilibrium of the game, and in this case $\underline{u}^d = \overline{u}^d$.*

Proof. By Lemma A.2, for any matrix $\mathbf{M}$ of order $|\mathcal{A}| \times |\mathcal{A}|$, we have either (1) $\exists \boldsymbol{x} \in \mathbb{R}^{|\mathcal{A}|}$, $\boldsymbol{x} \neq \mathbf{0}$ and $\boldsymbol{x} \succeq \mathbf{0}$, *s.t.* $\boldsymbol{x}^\top\mathbf{M} \succeq \mathbf{0}$, or (2) $\exists \boldsymbol{y} \in \mathbb{R}^{|\mathcal{A}|}$, $\boldsymbol{y} \neq \mathbf{0}$ and $\boldsymbol{y} \succeq \mathbf{0}$, *s.t.* $\mathbf{M}\boldsymbol{y} \preceq \mathbf{0}$. Take $\boldsymbol{s}^a = \frac{\boldsymbol{x}}{\sum_{i=1}^{|\mathcal{A}|} x_i}$, $\boldsymbol{s}^d = \frac{\boldsymbol{y}}{\sum_{i=1}^{|\mathcal{A}|} y_i}$, and we have

$\boldsymbol{s}^a, \boldsymbol{s}^d \in \mathcal{S}$, then either (1) $(\boldsymbol{s}^a)^\top \mathbf{M} \succeq \mathbf{0}$ or (2) $\mathbf{M}\boldsymbol{s}^d \preceq \mathbf{0}$. Take a matrix

$$\mathbf{M} = \mathbf{C} - \mathbf{M}^d = \begin{bmatrix} c & c & \cdots & c \\ c & c & \cdots & c \\ \vdots & \vdots & \ddots & \vdots \\ c & c & \cdots & c \end{bmatrix} - \mathbf{M}^d \tag{12}$$

where $c \in \mathbb{R}$ is an arbitrary constant, then there are two cases.

Case I. $(\boldsymbol{s}^a)^\top \mathbf{M} \succeq \mathbf{0}$, which implies $\forall \boldsymbol{s}^d \in \mathcal{S}$,

$$\begin{aligned} (\boldsymbol{s}^a)^\top \mathbf{M} \boldsymbol{s}^d &\geq 0 \\ (\boldsymbol{s}^a)^\top (\mathbf{C} - \mathbf{M}^d) \boldsymbol{s}^d &\geq 0 \\ (\boldsymbol{s}^a)^\top \mathbf{M}^d \boldsymbol{s}^d &\leq (\boldsymbol{s}^a)^\top \mathbf{C} \boldsymbol{s}^d \\ (\boldsymbol{s}^a)^\top \mathbf{M}^d \boldsymbol{s}^d &\leq (\boldsymbol{s}^a)^\top (\sum_{i=1}^{|\mathcal{A}|} c s_i^d \cdot \mathbf{1}) \\ (\boldsymbol{s}^a)^\top \mathbf{M}^d \boldsymbol{s}^d &\leq c \cdot \sum_{i=1}^{|\mathcal{A}|} s_i^a \\ (\boldsymbol{s}^a)^\top \mathbf{M}^d \boldsymbol{s}^d &\leq c \\ \max_{\boldsymbol{s}^d \in \mathcal{S}} (\boldsymbol{s}^a)^\top \mathbf{M}^d \boldsymbol{s}^d &\leq c \\ \min_{\boldsymbol{s}^a \in \mathcal{S}} \max_{\boldsymbol{s}^d \in \mathcal{S}} (\boldsymbol{s}^a)^\top \mathbf{M}^d \boldsymbol{s}^d &\leq c \\ \overline{u}^d &\leq c \end{aligned}$$

Case II. $\mathbf{M}\boldsymbol{s}^d \preceq \mathbf{0}$, which implies $\forall \boldsymbol{s}^a \in \mathcal{S}$,

$$\begin{aligned} (\boldsymbol{s}^a)^\top \mathbf{M} \boldsymbol{s}^d &\leq 0 \\ (\boldsymbol{s}^a)^\top (\mathbf{C} - \mathbf{M}^d) \boldsymbol{s}^d &\leq 0 \\ (\boldsymbol{s}^a)^\top \mathbf{M}^d \boldsymbol{s}^d &\geq (\boldsymbol{s}^a)^\top \mathbf{C} \boldsymbol{s}^d \\ (\boldsymbol{s}^a)^\top \mathbf{M}^d \boldsymbol{s}^d &\geq (\boldsymbol{s}^a)^\top (\sum_{i=1}^{|\mathcal{A}|} c s_i^d \cdot \mathbf{1}) \\ (\boldsymbol{s}^a)^\top \mathbf{M}^d \boldsymbol{s}^d &\geq c \cdot \sum_{i=1}^{|\mathcal{A}|} s_i^a \\ (\boldsymbol{s}^a)^\top \mathbf{M}^d \boldsymbol{s}^d &\geq c \\ \min_{\boldsymbol{s}^a \in \mathcal{S}} (\boldsymbol{s}^a)^\top \mathbf{M}^d \boldsymbol{s}^d &\geq c \\ \max_{\boldsymbol{s}^d \in \mathcal{S}} \min_{\boldsymbol{s}^a \in \mathcal{S}} (\boldsymbol{s}^a)^\top \mathbf{M}^d \boldsymbol{s}^d &\geq c \\ \underline{u}^d &\geq c \end{aligned}$$

Thus, we have either $\underline{u}^d \geq c$ or $\overline{u}^d \leq c$ and c is arbitrary. Considering that $\overline{u}^d \geq \underline{u}^d$, we have $\overline{u}^d = \underline{u}^d = u^*$. Therefore, the game has a value u^* satisfying Nash equilibrium.

Take $\boldsymbol{s}^{a*}, \boldsymbol{s}^{d*} \in \mathcal{S}$ *s.t.* $\min_{\boldsymbol{s}^a \in \mathcal{S}} (\boldsymbol{s}^{a*})^\top \mathbf{M}^d \boldsymbol{s}^d = \underline{u}^d$ and $\max_{\boldsymbol{s}^d \in \mathcal{S}} (\boldsymbol{s}^{a*})^\top \mathbf{M}^d \boldsymbol{s}^d = \overline{u}^d$, then we have

$$\underline{u}^d \leq (\boldsymbol{s}^{a*})^\top \mathbf{M}^d \boldsymbol{s}^{d*} \leq \overline{u}^d \tag{13}$$

therefore $(\boldsymbol{s}^{a*})^\top \mathbf{M}^d \boldsymbol{s}^{d*} = u^*$. By definition of $\underline{u}^d$ and $\overline{u}^d$ in Eqs. (8) and (9), we have $u^d(\boldsymbol{s}^{a*}, \boldsymbol{s}^d) \leq u^d(\boldsymbol{s}^{a*}, \boldsymbol{s}^{d*}) \leq u^d(\boldsymbol{s}^a, \boldsymbol{s}^{d*})$ for all $\boldsymbol{s}^a, \boldsymbol{s}^d \in \mathcal{S}$. This finishes the proof of Theorem A.3. □

A.2. Specific Method to Solve the Nash Equilibrium

In the implementation of NEED, we use linear programming to solve the Nash equilibrium of the attack-defense game and find the defender's best strategy. Specifically, we first transform the objective in Eq. (8) into

$$\begin{aligned} \min_{\boldsymbol{s}^d \in \mathbb{R}^{|\mathcal{A}|}} & -\underline{u}^d \\ \textit{s.t.} & -\hat{\mathbf{M}}^d \boldsymbol{s}^d - (-\underline{u}^d) \cdot \mathbf{1} \preceq \mathbf{0} \end{aligned}$$

which minimizes $-\underline{u}^d$.

Let $\boldsymbol{x}$ be a vector concatenating $\boldsymbol{s}^d$ and $[-\underline{u}^d]$, $\boldsymbol{x} = [s_1^d, \cdots, s_{|\mathcal{A}|}^d, -\underline{u}^d] \in \mathbb{R}^{|\mathcal{A}|+1}$, and we can formulate the linear program as follows:

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathbb{R}^{|\mathcal{A}|+1}} & \boldsymbol{c}^\top \boldsymbol{x} \\ \textit{s.t.} \quad & \mathbf{M}_{\text{ub}} \boldsymbol{x} \preceq \boldsymbol{b}_{\text{ub}} \\ & \mathbf{M}_{\text{eq}} \boldsymbol{x} = b_{\text{eq}} \\ & x_i \geq 0 \qquad \text{for } i \leq |\mathcal{A}| \end{aligned} \tag{14}$$

The coefficients of the linear program are defined by:

$$\begin{aligned} \boldsymbol{c} &= [0, \cdots, 0, 1]^\top & \boldsymbol{c} &\in \{0,1\}^{|\mathcal{A}|+1} \\ \mathbf{M}_{\text{ub}} &= \begin{bmatrix} -\hat{\mathbf{M}}_{11}^d & \cdots & -\hat{\mathbf{M}}_{1|\mathcal{A}|}^d & -1 \\ \vdots & \ddots & \vdots & \vdots \\ -\hat{\mathbf{M}}_{|\mathcal{A}|1}^d & \cdots & -\hat{\mathbf{M}}_{|\mathcal{A}||\mathcal{A}|}^d & -1 \end{bmatrix} & \mathbf{M}_{\text{ub}} &\in \mathbb{R}^{|\mathcal{A}| \times (|\mathcal{A}|+1)} \\ \boldsymbol{b}_{\text{ub}} &= [0, \cdots, 0]^\top & \boldsymbol{b}_{\text{ub}} &\in \{0\}^{|\mathcal{A}|} \\ \mathbf{M}_{\text{eq}} &= \begin{bmatrix} 1 & \cdots & 1 & 0 \end{bmatrix} & \mathbf{M}_{\text{eq}} &\in \{0,1\}^{1 \times (|\mathcal{A}|+1)} \\ b_{\text{eq}} &= 1 \end{aligned}$$

In the solution of this linear program $\boldsymbol{x}^* = [s_1^{d*}, \cdots, s_{|\mathcal{A}|}^{d*}, -\underline{u}^{d*}]^\top$, we have the best strategy of the defender $\boldsymbol{s}^{d*} = [s_1^{d*}, \cdots, s_{|\mathcal{A}|}^{d*}]^\top$. Similarly, we can solve the best strategy of the attacker $\boldsymbol{s}^{a*}$ by transforming Eq. (9) into exactly the same linear program in Eq. (14), but with a different $\boldsymbol{x} = [s_1^a, \cdots, s_{|\mathcal{A}|}^a, \overline{u}^d] \in \mathbb{R}^{|\mathcal{A}|+1}$ and different coefficients:

$$\begin{aligned} \boldsymbol{c} &= [0, \cdots, 0, 1]^\top & \boldsymbol{c} &\in \{0,1\}^{|\mathcal{A}|+1} \\ \mathbf{M}_{\text{ub}} &= \begin{bmatrix} (\hat{\mathbf{M}}^d)_{11}^\top & \cdots & (\hat{\mathbf{M}}^d)_{1|\mathcal{A}|}^\top & -1 \\ \vdots & \ddots & \vdots & \vdots \\ (\hat{\mathbf{M}}^d)_{|\mathcal{A}|1}^\top & \cdots & (\hat{\mathbf{M}}^d)_{|\mathcal{A}||\mathcal{A}|}^\top & -1 \end{bmatrix} & \mathbf{M}_{\text{ub}} &\in \mathbb{R}^{|\mathcal{A}| \times (|\mathcal{A}|+1)} \\ \boldsymbol{b}_{\text{ub}} &= [0, \cdots, 0]^\top & \boldsymbol{b}_{\text{ub}} &\in \{0\}^{|\mathcal{A}|} \\ \mathbf{M}_{\text{eq}} &= \begin{bmatrix} 1 & \cdots & 1 & 0 \end{bmatrix} & \mathbf{M}_{\text{eq}} &\in \{0,1\}^{1 \times (|\mathcal{A}|+1)} \\ b_{\text{eq}} &= 1 \end{aligned}$$

Table A1. The robust accuracy scores (%) on CIFAR-10 dataset obtained by different evaluation schemes on different network architectures. The lowest scores of each column are set in **bold**.

Method	Network	Evaluation	FGSM	PGD-20	PGD-100	EoT-PGD-20	VMI-FGSM	AutoAttack
Dynamic [3]	WideResNet-34-10 [46] (4 exits)	Single attack	57.75 ± 0.00	49.64 ± 0.04	47.90 ± 0.04	47.15 ± 0.06	48.77 ± 0.04	47.64 ± 0.08
		Average attack	60.42 ± 0.00	52.39 ± 0.05	49.43 ± 0.08	49.76 ± 0.08	51.19 ± 0.03	52.27 ± 0.20
		Max-average attack	62.06 ± 0.00	55.94 ± 0.09	53.73 ± 0.07	54.08 ± 0.12	55.35 ± 0.12	47.32 ± 0.22
		AIMER (ours)	**57.73** ± 0.00	**49.23** ± 0.08	**46.68** ± 0.04	**46.80** ± 0.10	**48.40** ± 0.04	**47.02** ± 0.09
Dynamic [3]	L2W-DEN [12] (5 exits)	Single attack	56.04 ± 0.00	51.24 ± 0.08	49.54 ± 0.05	49.90 ± 0.07	50.23 ± 0.08	63.33 ± 0.06
		Average attack	52.11 ± 0.00	46.65 ± 0.14	44.67 ± 0.08	44.97 ± 0.09	46.13 ± 0.06	65.04 ± 0.15
		Max-average attack	60.70 ± 0.00	54.81 ± 0.23	52.09 ± 0.10	52.85 ± 0.12	53.38 ± 0.05	63.12 ± 0.32
		AIMER (ours)	**51.48** ± 0.00	**44.07** ± 0.20	**41.86** ± 0.11	**42.32** ± 0.21	**43.05** ± 0.09	**62.15** ± 0.20
Dynamic [3]	RANet [43] (4 exits)	Single attack	50.94 ± 0.00	45.94 ± 0.06	44.01 ± 0.03	44.50 ± 0.10	44.78 ± 0.03	43.98 ± 0.05
		Average attack	50.51 ± 0.00	45.35 ± 0.09	43.50 ± 0.09	43.84 ± 0.08	44.71 ± 0.06	53.66 ± 0.09
		Max-average attack	55.47 ± 0.00	51.31 ± 0.10	49.55 ± 0.11	49.92 ± 0.22	50.06 ± 0.06	43.75 ± 0.15
		AIMER (ours)	**49.86** ± 0.00	**44.20** ± 0.06	**42.38** ± 0.08	**42.71** ± 0.13	**43.53** ± 0.08	**43.39** ± 0.08

Table A2. The robust accuracy scores (%) on **Tiny ImageNet** dataset obtained by different evaluation schemes. The lowest scores of each column are set in **bold**.

Method	Network	Evaluation	FGSM	PGD-20	PGD-100	EoT-PGD-20	VMI-FGSM
Dynamic [3]	MSDNet (5 exits)	Single attack	21.17 ± 0.00	19.67 ± 0.05	19.33 ± 0.04	19.30 ± 0.03	19.43 ± 0.02
		Average attack	12.99 ± 0.00	10.63 ± 0.04	10.36 ± 0.02	10.18 ± 0.02	10.49 ± 0.01
		Max-average attack	19.49 ± 0.00	18.31 ± 0.02	17.17 ± 0.02	17.58 ± 0.06	13.27 ± 0.03
		AIMER (ours)	**12.71** ± 0.00	**10.27** ± 0.04	**10.10** ± 0.03	**9.80** ± 0.05	**10.21** ± 0.03

Since there are readily available tools for solving the linear program to find the Nash equilibrium, we directly utilize the `Nashpy`[1] python package in our method.

B. Additional Experimental Results

In addition to the experiments in the main text, in this section, we present and analyze additional experimental results, providing more comprehensive evidence for the effectiveness of the methods proposed in this paper.

Evaluation with AIMER on more network architectures and datasets. We conduct the evaluation of AIMER and other schemes on three more multi-exit neural network architectures. In Table A1, we present the results on the multi-exit WideResNet-34-10, L2W-DEN, and RANet. These additional results consistently verify the effectiveness of AIMER, which evaluates relatively lower robustness scores than other three schemes. In Table A2, we evaluate MSDNet on Tiny ImageNet dataset, which has more samples, a larger number of classes and higher resolutions than CIFAR-10 and SVHN. The results show that our method can outperform other evaluation schemes on this dataset, exhibiting generalizability over both smaller and larger datasets.

[1] https://github.com/drvinceknight/Nashpy

Comparing AIMER with different single attacks. In Section 4.2, we construct an ad-hoc ResNet-18 with static inference strategy (always using the 3rd exit for inference). Our intention for this setup is to show that a mismatch happens in such a case and the robustness is severely overestimated using single attack. In Table A3, we extended the static inference experiment in Table 1 by showing all the possible single attacks using each of the exits (*e.g.*, "Single attack (exit-i)" denotes attacking the i-th exit). Note that the "Single attack (exit-3)" is exactly the case of A-D match. Overall, the results are consistent with the remarks in Section 3.2.

- When the results of "Single attack (exit-i)" ($i \neq 3$) are compared with "Single attack (exit-3)" ($E_a = \{3\}$ and $E_d = \{3\}$), the single attack-exit-3 accuracy scores are lower, which follows Remark 3.3;
- When the results of average attack ($E_a = \{1, 2, 3, 4\}$ and $E_d = \{3\}$) are compared with "Single attack (exit-3)" ($E_a = \{3\}$ and $E_d = \{3\}$), the "Single attack (exit-3)" accuracy scores are lower, which follows Remark 3.2.

Also, it can be noticed that AIMER has a very similar performance with the "Single attack (exit-i)" that makes a perfect match in attack and defense. This shows that the strategy of AIMER can always find the optimal choice in the static setting.

NEED performance on more network architectures and datasets. We compare the defense performance of NEED

Table A3. The robust accuracy scores (%) on **CIFAR-10** dataset obtained by different single attacks, average attack and AIMER. The lowest scores of each column are set in **bold**.

Method	Network	Evaluation	FGSM	PGD-20	PGD-100	EoT-PGD-20	VMI-FGSM	AutoAttack
Static (3/4)	ResNet-18 (4 exits)	Single attack (exit-1)	75.04 ± 0.00	75.43 ± 0.04	75.44 ± 0.07	75.35 ± 0.05	74.73 ± 0.04	80.15 ± 0.05
		Single attack (exit-2)	66.96 ± 0.00	65.76 ± 0.05	65.41 ± 0.06	65.36 ± 0.05	65.20 ± 0.06	74.28 ± 0.03
		Single attack (exit-3)	52.83 ± 0.00	45.86 ± 0.05	43.24 ± 0.03	43.64 ± 0.08	**45.10** ± 0.04	42.41 ± 0.02
		Single attack (exit-4)	60.29 ± 0.00	56.54 ± 0.08	54.12 ± 0.02	54.42 ± 0.06	55.30 ± 0.05	59.34 ± 0.02
		Average attack	56.54 ± 0.00	52.09 ± 0.04	50.51 ± 0.04	50.72 ± 0.04	51.38 ± 0.05	56.49 ± 0.03
		AIMER (ours)	**52.81** ± 0.00	**45.85** ± 0.05	**43.21** ± 0.01	**43.60** ± 0.03	**45.10** ± 0.03	**42.40** ± 0.04

Table A4. The Accuracy (%) of single-exit and different multi-exit **WideResNet-34-10** on **CIFAR-10** evaluated with AIMER. The best result of each column is set in **bold**.

Network	Method	Clean Acc.	Robust Acc.
WideResNet-34-10 (4 exits)	Single-exit	86.31 ± 0.00	43.78 ± 0.04
	Static	86.82 ± 0.00	49.37 ± 0.06
	Dynamic	86.85 ± 0.00	49.23 ± 0.12
	NEED	**87.90** ± 0.00	**50.78** ± 0.11

Table A5. The Accuracy (%) of two multi-exit neural networks on **CIFAR-10** with different inference strategies evaluated with AIMER. The best result of each column is set in **bold**.

Network	Method	Clean Acc.	Robust Acc.
L2W-DEN [12]	Static	**83.20** ± 0.00	32.84 ± 0.04
	Dynamic	81.64 ± 0.00	44.12 ± 0.08
	NEED	81.80 ± 0.05	**48.03** ± 0.10
RANet [43]	Static	**78.57** ± 0.00	38.03 ± 0.03
	Dynamic	78.11 ± 0.00	44.20 ± 0.15
	NEED	78.52 ± 0.02	**45.92** ± 0.21

on three additional architectures in Table A4 and A5. Compared with the baselines, NEED achieves a higher robustness under the strict evaluation of AIMER. Additionally, Table A8 shows the defense performance of NEED combined with different adversarial training methods using ResNet-18 backbone on SVHN dataset. In most cases, NEED outperforms the single-exit baselines by a noticeable margin. Especially, in AutoAttack, NEED can improve the robust score by around 5% maximally. We suppose that this might be attributed to the significant impact of the maximized A-D mismatch on the decision-based algorithm in AutoAttack, causing the unstable adversarial examples it generates to easily become ineffective in a different ensemble of exits for inference. Meanwhile, we also notice that the performance of NEED lags behind single-exit networks when using FAT adversarial training. This is probably caused by the complexity of multi-exit neural networks which makes it more challenging to transfer the robustness obtained from iterative attacks during training to single-step attacks when using certain adversarial training methods.

Table A6. Accuracy (%) under **LAFIT** attack on CIFAR-10. The setup of different networks are consistent with those in Table 1 and A1. The best result of each column is set in **bold**.

Evaluation	ResNet-18	MSDNet	ViT
Single attack	62.62 ± 0.10	56.51 ± 0.05	56.53 ± 0.03
Average attack	62.80 ± 0.07	42.91 ± 0.15	58.85 ± 0.09
Max-average attack	57.20 ± 0.05	55.40 ± 0.08	50.36 ± 0.05
AIMER (ours)	**45.27** ± 0.02	**42.89** ± 0.08	**50.01** ± 0.08

Table A7. Comparison between the AIMER evaluation scheme and an exit-wise EoT evaluation scheme from the dimensions of PGD-20 robust accuracy and online computational cost per sample. Experiments are conducted on **ResNet-18** network architecture and **CIFAR-10** dataset.

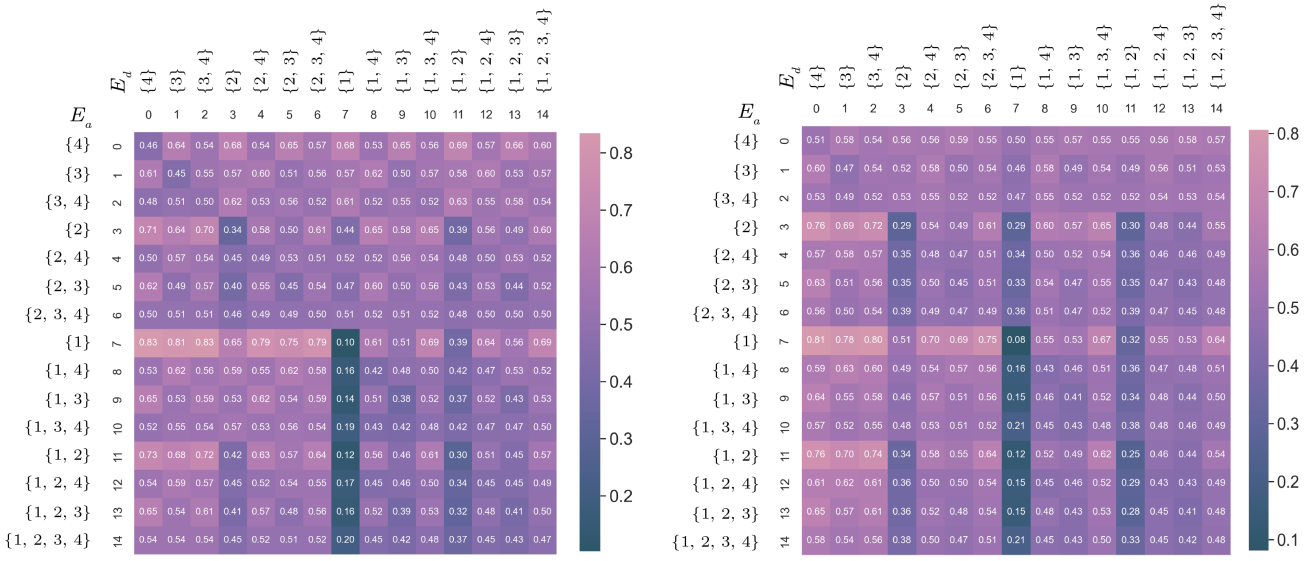
Network	Method	Robust Acc. (%)	Cost (ms)
ResNet-18 (4 exits)	**AIMER** (ours)	45.22 ± 0.04	3.88 $\times 10^0$
	EoT (1-iter.)	50.93 ± 0.05	3.85 $\times 10^0$
	EoT (2-iter.)	48.73 ± 0.04	7.73 $\times 10^0$
	EoT (5-iter.)	47.37 ± 0.06	1.93 $\times 10^1$
	EoT (10-iter.)	45.79 ± 0.05	3.87 $\times 10^1$
	EoT (20-iter.)	45.12 ± 0.06	7.78 $\times 10^1$

Evaluation with AIMER using latent feature attacks. Latent feature attacks, such as LAFIT [45], have a close connection with multi-exit neural networks in that they also utilize early exit results to optimize the adversarial examples. In Table A6, we attempt to further enhance such attacks with AIMER. For a n-exit network, we use the $(n-2)$-th exit to acquire latent features and replace the main-exit results with different evaluation schemes. While the results also shows consistent effectiveness of AIMER, LAFIT does not necessarily perform better than other attacks like PGD-100. We assume that this might be caused by the lack of a flexible strategy in choosing the latent features, which might lead to severe A-D mismatch. It remains a challenging yet promising future direction to study how to better combine AIMER with this category of adversarial attacks.

Visualization of the approximated payoff matrices. Figure A1, A4 and A5 plot the heatmaps of the approxi-

Table A8. Accuracy (%) under different attacks when combining NEED with AT methods on the **ResNet-18** model and **SVHN** dataset. Better results are set in **bold**.

Method	Clean	FGSM	PGD-20	PGD-100	EoT-PGD-20	VMI-FGSM	AutoAttack
Standard	95.46 ± 0.00	21.55 ± 0.00	0.78 ± 0.01	0.18 ± 0.02	0.30 ± 0.01	0.44 ± 0.01	0.18 ± 0.01
Standard + NEED	**95.80** ± 0.04	**26.23** ± 0.12	**0.79** ± 0.06	**0.54** ± 0.06	**0.42** ± 0.04	**0.46** ± 0.04	**1.64** ± 0.10
PGD-AT [22]	92.17 ± 0.00	72.67 ± 0.00	54.20 ± 0.05	44.64 ± 0.04	51.58 ± 0.03	47.00 ± 0.01	25.48 ± 0.03
PGD-AT + NEED	**93.34** ± 0.05	**75.31** ± 0.11	**55.03** ± 0.17	**46.50** ± 0.12	**53.11** ± 0.10	**54.15** ± 0.09	**30.10** ± 0.21
TRADES [47]	89.52 ± 0.00	73.95 ± 0.00	54.09 ± 0.05	50.28 ± 0.03	52.14 ± 0.01	51.53 ± 0.03	47.45 ± 0.04
TRADES + NEED	**93.29** ± 0.05	**74.02** ± 0.10	**55.44** ± 0.16	**50.99** ± 0.09	**52.59** ± 0.18	**52.04** ± 0.22	**50.12** ± 0.30
MART [38]	91.32 ± 0.00	72.71 ± 0.00	49.11 ± 0.01	40.66 ± 0.05	42.25 ± 0.06	40.08 ± 0.07	33.99 ± 0.01
MART + NEED	**92.16** ± 0.16	**74.34** ± 0.11	**50.43** ± 0.33	**42.89** ± 0.15	**46.15** ± 0.12	**46.11** ± 0.20	**37.91** ± 0.13
FAT [48]	**93.00** ± 0.00	**56.87** ± 0.00	48.04 ± 0.05	46.80 ± 0.05	45.18 ± 0.02	47.05 ± 0.03	43.85 ± 0.02
FAT + NEED	90.49 ± 0.03	56.18 ± 0.08	**49.01** ± 0.04	**47.29** ± 0.08	**46.90** ± 0.07	**48.80** ± 0.09	**44.76** ± 0.12



(a) SVHN (b) CIFAR-10

Figure A1. The defender's approximated payoff matrix on the **ResNet-18**. The values in the matrix are estimated with a 5-batch evaluation under PGD-20 attack.

mated payoff matrix of different network structures on different datasets. Among them, Figure A1a and A1b are generated on a 4-exit ResNet-18 network, while Figure A4 and A5 are generated on a 5-exit VGG-16 network. Like Figure 3, the A-D mismatch can also be easily observed in these figures, *i.e.*, the lowest value of each column is the diagonal element. Although in extremely special cases, we can observe some counterexamples, it may be explained by the insufficient accuracy in estimating the payoff matrix, and this anomaly can be avoided when we test with larger datasets.

C. Further Discussion

C.1. Is EoT All You Need?

Expectation over Transformation (EoT) [1] is a widely used technique to counteract the randomness in the robustness evaluation of networks, which seems to be an alternative to AIMER. However, we argue that AIMER has its unique advantage, *i.e.*, the *online* computational cost is significantly lower than EoT. In Table A7, we compare the experimental results of simply using EoT and applying AIMER in evaluating a multi-exit neural network with a dynamic inference strategy. It can be observed that when EoT struggles to catch up with the performance of AIMER, its online cost sharply rises. In contrast, AIMER achieves a much higher online efficiency, while it only needs to calculate the offline

payoff matrix beforehand. This solution is rather effective when encountered with heavy evaluation tasks, *e.g.*, when the attack settings include AutoAttack or multiple attacks.

C.2. A Dialectical Perspective on A-D Mismatch

This paper discovers the A-D mismatch phenomenon in the robustness evaluation of multi-exit neural networks. On one hand, it leads to an overestimation of the network's robustness when evaluating with a fixed exit, resulting in measured results higher than the network's true robustness (Section 3.2). Therefore, it can be considered a factor hindering the robustness evaluation process. However, on the other hand, for networks that tend to use different exits for inference (including dynamic and random inference), A-D mismatch is an unavoidable phenomenon. In other words, the attacker can only minimize the mismatch through methods like AIMER (Section 4.2) but cannot completely eliminate it. In such cases, the defender can still apply methods like NEED to enhance the mismatch rate for more robustness (Section 4.3).

We believe that whether A-D mismatch is good or bad is still an open question. Due to the uncertainty in the choices made by both the attacker and defender, the additional robustness introduced by mismatch cannot (in most cases) be entirely disentangled from the network's intrinsic robustness. Therefore, we can even consider it as a part of the multi-exit adversarial defense methods. However, considering that excessive mismatch can have a significant impact on the results, efforts should be made to minimize its effects during evaluation. This will enable researchers in the field of adversarial defense to have a more accurate understanding of the true effectiveness of their defense methods.

D. Inference of Multi-Exit Neural Networks

In this paper, the inference strategies of multi-exit neural networks are divided into the following three categories:

Static inference makes predictions with fixed exit(s). The *any-time prediction* strategy (i.e., using a fixed exit for inference) in previous literature [10, 12, 16, 43] belongs to this category. We formulate this inference as follows:

$$f^{\text{static}}_{\theta,E_d}(x) = \frac{1}{|E_d|}\sum_{i\in E_d} f_{\theta_i}(x) \tag{15}$$

Dynamic inference is another commonly used inference strategy of multi-exit neural networks. It sequentially decides whether to break from the current exit according to a confidence threshold. The *budgeted prediction* strategy used in previous literature [10, 12, 16, 43] belongs to this category. We formulate it as

$$f^{\text{dynam}}_{\theta}(x) = \begin{cases} f_{\theta_1}(x), & \text{if } \phi_1(f_{\theta_1}(x)) \\ f_{\theta_i}(x), & \text{if } \bigwedge_{j=1}^{i-1}\neg\phi_j(f_{\theta_j}(x)) \wedge \phi_i(f_{\theta_i}(x)), \\ & 2 \le i \le L-1 \\ f_{\theta_L}(x), & \text{otherwise} \end{cases}, \tag{16}$$

where $\phi_i : \mathcal{Y} \to \{0, 1\}$ is a boolean-valued function that evaluates whether the confidence of the prediction qualifies a certain condition, *e.g.*, the maximal entry of the prediction is larger than a threshold or the entropy of the prediction is smaller than a threshold.

Random inference is an ad-hoc inference strategy in our paper as the defender's counterpart of partial attack, which can increase the difficulty of attack by making the A-D match more difficult. It is formulated as:

$$f^{\text{random}}_{\theta,\mathbf{s}^d}(x) = f^{\text{static}}_{\theta,\text{random}(\mathcal{A},\mathbf{s}^d)}(x) \tag{17}$$

Another important function of this type of inference is to approximate the model's behavior of dynamic inference. Note that in different dynamic inference implementations, *the function ϕ is not unique*, making it very hard to establish a universal model of the network's behavior under attack. Hence, we utilize the probabilistic behavior of the network when it is attacked, taking a random inference model as a surrogate for the dynamic inference model in AIMER.

E. A Visual Comparison Between AIMER and Previous Evaluation Schemes

Figure A2 makes a comparison between AIMER and previous evaluation schemes, from which we can easily figure out their difference in the choice of target exits and strategies to find an optimal scheme for attack.

In a single attack (Figure A2(a)), the attacker only targets one of all the exits for generating adversarial examples, while the rest of the exits are overlooked. This attack can be easily detoured when the defender uses an exit that is not targeted by the attacker. In contrast, an average attack (Figure A2(b)) goes to the other extreme by averaging the adversarial effect produced on each exit. Although it considers each exit, its impact on each individual exit is too dispersed, significantly weakening the attack effect. This can lead to an overestimation of robustness when the defender relies solely on one exit for inference. The max-average attack (Figure A2(c)) addresses this issue by using the "best" choice among all single attacks to generate adversarial examples. However, this approach is still limited by the best performance of single attacks. Also, the objective of max-average cannot always ensure it finds the single attack that minimizes the evaluated robustness (see Section 4.2).

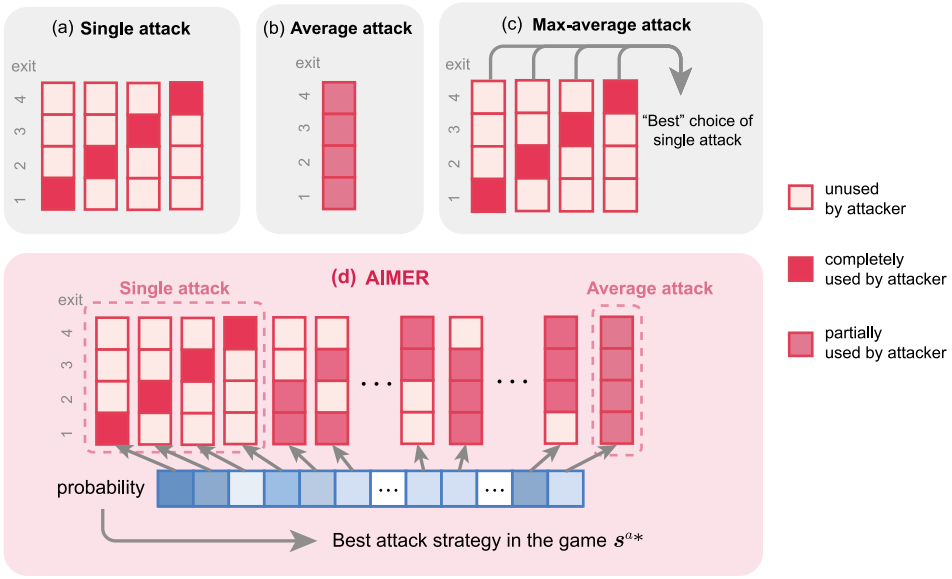


Figure A2. A visual comparison between AIMER and previous evaluation schemes.

AIMER (Figure A2(d)) differs from the aforementioned attacks in that, set in the context of a two-player zero-sum game, it not only enumerates all possible ensembles of exits but also spans these ensembles as a basis in a probability space. Within this space, it searches for the optimal attack strategy. Therefore, compared to the previous three types of attacks, AIMER is capable of finding a more effective solution for the attack that optimizes the effect of the attack over a broader range.

F. Implementation Details

In this section, we elaborate on the detailed implementation of this paper. First, we explain the algorithmic process of AIMER; then, we analyze the time complexity of the payoff matrix approximation step and test its dynamic performance to decide the optimal parameter; thereafter, we further demonstrate the details in the experiments, including the implementation of multi-exit neural networks, the attack algorithms used in the evaluation, the experimental environment and the setting of hyperparameters.

F.1. The Algorithm of AIMER

We provide a detailed introduction to the algorithmic process of AIMER. As shown in Algorithm 1, we first initialize the defender's strategy according to the type of inference (static, random, or dynamic) used by the target multi-exit neural network. Next, we conduct a small-scale matrix test on the target multi-exit neural network, traversing all combinations of attack and inference exit ensembles and recording the accuracy for each scenario to estimate the defender's approximate payoff matrix. Then, we use the estimated payoff matrix and the defender's strategy to calculate the best strategy for the attacker in the game. Finally, we use the partial attack to implement the attacker's strategy, generating AIMER adversarial samples for robustness evaluation.

Algorithm 1 Adaptive Evaluation of Multi-Exit Robustness (AIMER)

Input: multi-exit neural network f_θ, inference type t, clean data x, ground-truth labels y, exit ensemble for static inference E_d, probability vector for random inference $\boldsymbol{p}^d$, batch size b, number of batches for quick test m

Output: Adversarial data for AIMER evaluation $x^{\text{adv}}_{\text{AIMER}}$

1: Initialize $\boldsymbol{s}^d \leftarrow$ INIT_DEF(f_θ, t, x, y, E_d, $\boldsymbol{p}^d$, b, m) ▷ Algorithm 2

2: Initialize $\hat{\mathbf{M}}^d \leftarrow$APPROXIMATE_PAYOFF($f_\theta$, t, x, y, b, m) ▷ Algorithm 3

3: Initialize $\boldsymbol{s}^{a*} \leftarrow$ best_attacker_strategy($\hat{\mathbf{M}}^d$, $\boldsymbol{s}^d$) ▷ Eq. (7)

4: Initialize $x^{\text{adv}}_{\text{AIMER}} \leftarrow$ partial_attack(f_θ, x, y, $\boldsymbol{s}^{a*}$) ▷ Eq. (4)

5: **return** $x^{\text{adv}}_{\text{AIMER}}$

F.2. Time Complexity and Dynamic Performance in Approximation of Payoff Matrix

In the algorithm of AIMER, we use an approximated payoff matrix rather than a payoff matrix tested on the complete dataset to reduce the computational cost. Since the network performance on a random subset of a dataset can

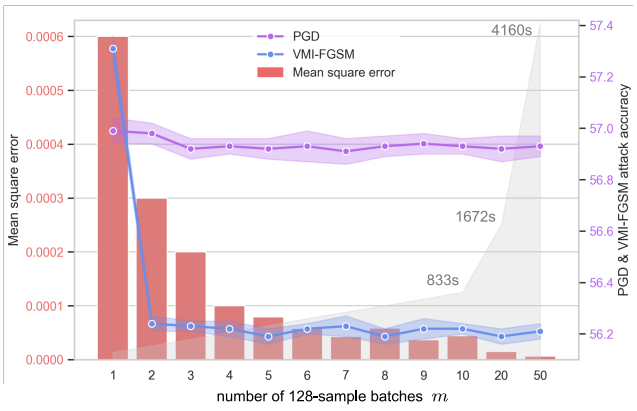


Figure A3. Dynamic performance of AIMER using different numbers of 128-sample batches to approximate the payoff matrix. The gray area in the figure represents the time consumption in the test.

Algorithm 2 Initialization of the defender's strategy (INIT_DEF)

Input: multi-exit neural network f_θ, inference type t, clean data x, ground-truth labels y, exit ensemble for static inference E_d, probability vector for random inference $\boldsymbol{p}^d$, batch size b, number of batches for quick test m
Output: The defender's strategy $\boldsymbol{s}^d$

if $t =$ `'static'` **then**
  ▷ Initialize the defender's strategy for static inference
  Initialize $\boldsymbol{s}^d \leftarrow [0 \text{ for each } E \in \mathcal{A}]$
  $s^d_{E_d} \leftarrow 1$
end if
if $t =$ `'random'` **then** ▷ Initialize the defender's random inference strategy
  Initialize $\boldsymbol{s}^d \leftarrow \boldsymbol{p}^d$
end if
if $t =$ `'dynamic'` **then** ▷ Approximate the defender's dynamic inference strategy
  Initialize $\boldsymbol{s}^d \leftarrow [0 \text{ for each } E \in \mathcal{A}]$
  for $i = 1$ **to** m **do** ▷ Use a small m that is trivial compared with the complete dataset
    for $j = 1$ **to** b **do**
      $x^{\text{adv}}_{\text{avg}} \leftarrow \text{average\_attack}(f_\theta, x_{ij}, y_{ij})$
      $E \leftarrow \text{exit\_from\_ensemble}(f_\theta, x^{\text{adv}}_{\text{avg}})$
        ▷ Detect which exit ensemble is used
      $s^d_E \leftarrow s^d_E + 1/mb$
    end for
  end for
end if
return $\boldsymbol{s}^d$

be highly consistent with that of on the complete dataset, we intuitively use a small number of data batches to calculate each element in the matrix. Suppose that we use m batches of data (with a batch size b) to test each accuracy score (*i.e.*, the payoff function value) in the matrix, and the matrix is of size $|\mathcal{A}| \times |\mathcal{A}|$, where $|\mathcal{A}| = 2^L - 1$. The time complexity of calculating the approximated payoff matrix is $\mathcal{O}(m \times b \times 2^L)$, which is exponential due to the enumeration of all possible ensemble of exits. In such a case, we have no choice but to reduce the computational cost on other dimensions like m.

Algorithm 3 Calculation of the approximated payoff matrix for the defender (APPROXIMATE_PAYOFF)

Input: multi-exit neural network f_θ, clean data x, ground-truth labels y, batch size b, number of batches for quick test m
Output: approximated payoff matrix for the defender $\hat{\mathbf{M}}^d$

Initialize $\hat{\mathbf{M}}^d \leftarrow [[0 \text{ for each } E \in \mathcal{A}] \text{ for each } E \in \mathcal{A}]$
for each $E_a \in \mathcal{A}$ **do**
  for each $E_d \in \mathcal{A}$ **do**
    for $i = 1$ **to** m **do** ▷ Use a small m that is trivial compared with the complete dataset
      for $j = 1$ **to** b **do**
        $x^{\text{adv}}_{\text{par}} \leftarrow \text{partial\_attack}(f_\theta, x_{ij}, y_{ij}, E_a)$
        $s^d_E \leftarrow s^d_E + 1/mb$
        $\hat{y} \leftarrow \frac{1}{|E_d|} \sum_{k \in E_d} (f_{\theta_k}(x^{\text{adv}}_{\text{par}}))$
          ▷ Inference with E_d using Eq. (15)
        if $\hat{y} = y_{ij}$ **then**
          $\hat{\mathbf{M}}^d_{E_a, E_d} \leftarrow \hat{\mathbf{M}}^d_{E_a, E_d} + 1/mb$
        end if
      end for
    end for
  end for
end for
return $\hat{\mathbf{M}}^d$

To identify a proper m that allows both efficient calculation of the payoff matrix and the effective implementation of AIMER, we conduct the following dynamic performance test. We evaluate the random inference (same setting as Section 4.2) with AIMER using different m values and monitor 4 factors: (1) the time required to calculate the payoff matrix, (2) the mean square error between the approximated matrix and an accurate matrix, (3) the accuracy under PGD attack and (4) the accuracy under VMI-FGSM attack. In Figure A3, it can be noticed that when m is larger than 2, the accuracy scores are not sensitive to the choice of m; also, the mean square error of the approximated matrix decreases sharply until $m = 5$. To achieve a balance between computational efficiency, error in approximation and accuracy, we finally choose $m = 5$ to keep the mean square error under 0.0001.

F.3. Multi-Exit Neural Network Architectures

We employ 7 multi-exit neural network architectures in our experiments, 4 of them modified from common single-exit network architectures (ResNet-18 [14], VGG-16 [31], WideResNet-34-10 [46], ViT-B/16 [7]) and 3 of them designed by previous researchers (MSDNet [17], L2W-DEN [12] and RANet [43]).

Multi-exit ResNet-18. We modify ResNet-18 into a 4-exit network by inserting a simple branch classifier between each two blocks of the residual network, and 3 classifiers are inserted in total. The structure of the three branch classifiers can be expressed in PyTorch pseudocode: `[torch.nn.AdaptiveAvgPool2d((2,2)), torch.nn.Linear(4*c, c), torch.nn.ReLU(), torch.nn.Dropout(), torch.nn.Linear(c, num_classes)]`. The parameter `c` is set to $[160, 320, 640]$ for each branch classifier respectively and `num_classes` is the number of classes in the dataset.

Multi-exit WideResNet-34-10. We modify WideResNet-34-10 into a 4-exit network by inserting a simple branch classifier between each two blocks, and 3 classifiers are inserted in total. The structure of the three branch classifiers can be expressed in PyTorch pseudocode: `[torch.nn.AdaptiveAvgPool2d((2,2)), torch.nn.Linear(4*c, c), torch.nn.ReLU(), torch.nn.Dropout(), torch.nn.Linear(c, num_classes)]`. The parameter `c` is set to $[64, 128, 256]$ for each branch classifier respectively and `num_classes` is the number of classes in the dataset.

Multi-exit VGG-16. We modify VGG-16 into a 5-exit network by inserting a simple branch classifier between each two blocks of the VGGNet, and 4 classifiers are inserted in total. The structure of the four branch classifiers can be expressed in PyTorch pseudocode: `[torch.nn.AdaptiveAvgPool2d((1,1)), torch.nn.Linear(c, c), torch.nn.ReLU(), torch.nn.Dropout(), torch.nn.Linear(c, num_classes)]`. The parameter `c` is set to $[64, 128, 256, 512]$ for each classifier respectively and `num_classes` is the number of classes in the dataset.

Multi-exit ViT-B/16. We modify ViT-B/16 into a 4-exit network by inserting a simple branch classifier between each two blocks and 3 classifiers are inserted in total. The structure of each inserted classifier is in line with that of the default classifier head in ViT-B/16.

MSDNet. MSDNet, or Multi-Scale Dense Networks, is a convolutional neural network architecture designed for efficient and scalable image classification. In this paper, we mainly utilize its feature of multi-exit inference. Our experiments are based on the official implementation of MSDNet[2]. We build the network architecture with the following settings:

- For CIFAR-10: `nBlocks=5, nChannels=64, base=4, stepmode='even', step=2, growthRate=6, grFactor='1-2-4', prune='max', bnFactor='1-2-4', bottleneck=True, reduction=0.5`;
- For Tiny ImageNet: `nBlocks=5, nChannels=32, base=4, stepmode='even', step=4, growthRate=16, grFactor='1-2-4-4', prune='max', bnFactor='1-2-4-4', bottleneck=True, reduction=0.5`.

L2W-DEN. L2W-DEN is an enhanced multi-exit architecture that better trains the network by reweighting the samples. We use the official code of L2W-DEN[3] and base this architecture on the MSDNet with the following setting: `nBlocks=5, nChannels=16, base=1, stepmode='lin_grow', step=2, growthRate=6, grFactor='1-2-4', prune='max', bnFactor='1-2-4', bottleneck=True, reduction=0.5`.

RANet. RANet is a resolution adaptive multi-exit neural network architecture that utilizes spatial redundancy of input images. We conduct our experiments on this network architecture using its official implementation[4] with the following setting: `nBlocks=2, block_step=2, nChannels=16, base=1, stepmode='even', step=4, growthRate=6, scale_list='1-2-3-3', grFactor='4-2-1-1', prune='max', bnFactor='4-2-1-1', bottleneck=True, reduction=0.5, compress_factor=0.25`.

F.4. Attack Algorithms Applied in Experiments

In our experiments, we apply 6 attack algorithms in total to verify the effectiveness of AIMER and NEED. A brief introduction to these attacks and reasons for choosing them are stated below.

Fast Gradient Signed Method (FGSM) attack [32] is known for its simplicity and computational efficiency. It generates adversarial examples in a single step by perturbing the original input image. This perturbation is calculated

[2] https://github.com/kalviny/MSDNet-PyTorch
[3] https://github.com/LeapLabTHU/L2W-DEN
[4] https://github.com/yangle15/RANet-pytorch

by taking the sign of the gradients of the loss with respect to the input image and then multiplying it by a small ϵ. This creates an adversarial image that is very close to the original but enough to deceive the model.

Projected Gradient Descent (PGD) attack [22] is an iterative adversarial attack method that enhances the effectiveness of perturbations by applying small changes to the input data across multiple steps, each time adjusting the perturbation to maximize the model's prediction error. Unlike simpler, single-step methods like FGSM, PGD is more thorough and often more successful at deceiving models, especially those with defenses against adversarial examples, making it a robust tool for evaluating model security. In our experiments, we employ FGSM and PGD, two simple yet classic attack algorithms as a reference benchmark for testing the robustness of defense methods.

Expectation over Transformation (EoT) PGD [1] is used for crafting more effective adversarial examples against models that employ randomized defense strategies. It takes into account the randomness in these defenses by averaging the gradients of the model's output with respect to the input over multiple transformations. This approach ensures that the adversarial perturbations are robust to the random transformations applied as a part of the defense mechanism, thereby increasing the likelihood of successfully deceiving the model even when it employs randomization as a countermeasure to adversarial attacks. Considering the possible randomness in the strategy of NEED, we employ the EoT-enhanced PGD algorithm in our experiments as a stronger attack.

Variance Tuning (VMI-FGSM) [36] is proposed to enhance the class of iterative gradient-based attack methods and improve their attack transferability. In this paper, we consider this attack algorithm for the multiple exits to be partially heterogeneous network architectures, and by improving the transferability, we aim to generate more universal adversarial examples for all the network exits.

AutoAttack [4] is an ensemble of multiple adversarial attack methods, designed to automatically and robustly evaluate the resilience of neural networks against adversarial examples. It combines several state-of-the-art attack techniques, including gradient-based and decision-based attacks, ensuring a comprehensive assessment by exploiting different aspects of model vulnerabilities, often without requiring any hyperparameter tuning. We employ AutoAttack in our experiments as a comprehensive indicator for the adversarial robustness of multi-exit neural networks.

LAFIT [45] attack pierces through neural networks and generates strong adversarial examples with the help of latent features. We choose this attack for its close connection with multi-exit neural networks in utilizing early exit results to optimize the adversarial examples.

To implement these algorithms, we refer to the code of `torchattacks`[5] Python package and their official implementation[6], and further add support for multi-exit neural network attacks, enabling the application of partial attacks in various algorithms.

F.5. Environment and Hyperparameter Settings

Experimental environment. Our experiments are conducted in Python 3.8 environment on single NVIDIA RTX A6000 GPU. We use the deep learning framework PyTorch 1.13 to build all the neural network models and carry out training and testing.

Model training. We train all the networks with different architectures for 100 epochs. We optimize with the SGD optimizer, setting the momentum to 0.9. We adjust the learning rate α of training with a dynamic scheme by initializing $\alpha = 0.1$ (for some adversarial training methods with difficulty in convergence, we set $\alpha = 0.01$ or even smaller) and reducing it to 0.1 and 0.01 times the original value at the epochs 75 and 90 respectively.

Dynamic inference. To implement the dynamic inference of multi-exit neural networks, we refer to the official source code of [17]. Specifically, it sets a threshold value for the unnormalized maximum logits at each exit, and sequentially (from earlier exits to later exits) decides whether to finish the inference from the current exit.

On different network architectures and different datasets, we apply different thresholds for them following [16]. In multi-exit ResNet-18, we set the thresholds to $[6.33, 5.62, 4.85, -10^8]$ for CIFAR-10 and $[7.86, 9.05, 7.54, -10^8]$ for SVHN; in multi-exit VGG-16, we set the thresholds to $[4.25, 6.33, 6.36, 3.76, -10^8]$ for CIFAR-10 and $[3.17, 8.66, 7.69, 7.67, -10^8]$ for SVHN; in WideResNet-34-10, we set the thresholds to $[13.03, 9.46, 9.70, -10^8]$ for CIFAR-10; in MSDNet, we set the thresholds to $[11.31, 9.48, 7.27, 5.46, -10^8]$ for CIFAR-10, and $[5.83, 4.43, 4.61, 3.83, -10^8]$ for Tiny ImageNet; in L2W-DEN, we set the thresholds to $[7.92, 6.63, 5.28, 3.66, -10^8]$ for CIFAR-10; in RANet, we set the thresholds to $[8.35, 6.10, 3.66, -10^8]$ for CIFAR-10.

[5] https://github.com/Harry24k/adversarial-attacks-pytorch

[6] https://github.com/lafeat/lafeat

Table A9. Comparison in methodology between our work and Meunier et al. [24].

Method	**What** (is it)?	**When** (to apply it)?	**Where** (to apply it)?
AIMER (ours)	robustness evaluation scheme	test phase	multi-exit neural networks
NEED (ours)	defense (aggregation strategy)	test phase	multi-exit neural networks
Meunier et al. [24]	defense (adversarial training)	training phase	randomized classifiers

G. Difference From Previous Work of Adversarial Game

In this section, we compare this paper and the most related work [24] in detail and clarify the stark difference between them. Overall, [24] answers two questions for randomized classifiers: (1) whether the attack and defense of randomized classifiers can always reach a Nash Equilibrium; (2) How to design an algorithm for learning an optimally robust randomized classifier. Albeit game theory is applied to study adversarial robustness in both works, we argue that the **motivation, purpose, methodology, and design of experiments** are essentially different from our paper.

- **Motivation.** The critical A-D mismatch phenomenon is only specified in our work, which is a new discovery that motivates us to carry out our research.
- **Purpose.** Generally speaking, our paper is devoted to tackling the widespread overestimation of the adversarial robustness of multi-exit neural networks. Differently, [24] focuses on the theoretical part and is dedicated to showing that the Nash Equilibrium in the adversarial examples game can be reached.
- **Methodology.** The methods proposed in two studies differ in terms of what they are, when and where to apply them. Table A9 clearly compares their differences.
- **Design of experiments.** In our paper, we have validated the effectiveness of our method in a broader range. Our experiment includes more attack algorithms (FGSM, PGD, EoT-PGD, VMI-FGSM, AutoAttack) to verify the generalizability of our methods over varied adversarial settings. Also, we have included more network architectures to validate if our methods can work on different multi-exit neural networks.

Additionally, we believe our paper better bridges the gap between theory and application. In [24], though the authors thoroughly study the properties of the adversarial game, they claim that the algorithms "are not easily practicable in the case of deep learning". In contrast, our paper identifies a direct application in multi-exit deep neural networks and innovatively applies the theoretical results in both testing and improving the robustness of multi-exit neural networks.

H. Limitations and Future Work

In this paper, we propose an effective evaluation scheme AIMER for measuring the adversarial robustness of multi-exit neural networks, and a corresponding defense method NEED that counteracts the attack strategy of AIMER to improve the additional robustness of the network. However, there are still many limitations in our current work, which deserve further in-depth study in the future work.

Approximation of the payoff matrix. In AIMER, we simply use an approximated matrix as a surrogate for the true payoff matrix in the game. This may raise concerns of the following two aspects:

- The imprecision in the estimated values can lead to unstable results. While we cannot obtain an exact payoff matrix and resorting to approximation is a necessity, the resulting inaccuracy may potentially lead to unexpected outcomes. For example, some counterexamples of the A-D mismatch phenomenon may appear in certain situations, where AIMER might deviate from the best strategy.
- Estimating the payoff matrix comes with additional computational costs. To obtain the estimation of payoffs, matrix testing of the performance of the network for various attack scenarios is required during evaluation, enumerating the estimated values for each type of partial attack and defense. The computational cost of this process is dependent on two factors: (1) the size of the action spaces for both the attacker and defender and (2) the scale of testing for each element in the matrix. When the number of network exits increases, the size of action space also increases (exponentially), resulting in a significant computational burden and thus limiting the feasibility of AIMER for cases with more network exits. Similarly, when higher estimation precision is needed for the payoff matrix, the testing scale for each matrix element also increases, posing an inevitable trade-off between cost and precision.

Nevertheless, we have illustrated that AIMER can effectively mitigate A-D mismatch and minimize the additional robustness using an approximation. Future work can focus on seeking a more precise measurement of the payoff matrix, as well as reducing the computational cost in testing the multi-exit properties to acquire the matrix. We believe that with breakthroughs in these directions, AIMER can achieve a more efficient and accurate evaluation of the intrinsic adversarial robustness of multi-exit neural networks.

Using random inference as a surrogate for dynamic inference. The overall framework of this paper is estab-

lished on the game theory, in which probabilistic behaviors can be better modeled than other behaviors with complex mechanisms (*e.g.* dynamic inference with confidence thresholds). Hence, we opt to simplify the process of dynamic inference, *i.e.*, instead of setting complex heuristic rules or considering limited possible cases, we view from the perspective of the attacker and treat the network's choice of exits as a probabilistic behavior. Although extensive experiments on different architectures can verify the effectiveness of our method on networks with dynamic inference, the mechanism of dynamic inference only has a surrogate in the current framework, and more rigorous theoretical explanations is left to be made in future work.

The metric for measuring A-D mismatch. In the experiments section, we introduce the mismatch rate as a quantitative metric to measure the extent of A-D mismatch. Although this metric is sufficient for demonstrating the effectiveness of our method, we have identified certain limitations with it. As shown in Figure 6, the distribution of data points on the mismatch rate axis exhibits a high variance, indicating that our measurement of A-D mismatch is not very accurate. Upon reflection, we realize that the mismatch rate metric does not take into account the differences between different exits but treats each exit equally. The issue with this approach is that if the classifiers of exit have varying discriminative power, these differences cannot be reflected in the mismatch rate, ultimately leading to increased errors. We expect in future work, more reasonable metrics for measuring mismatch can be proposed to push forward the study of the adversarial robustness of multi-exit neural networks.

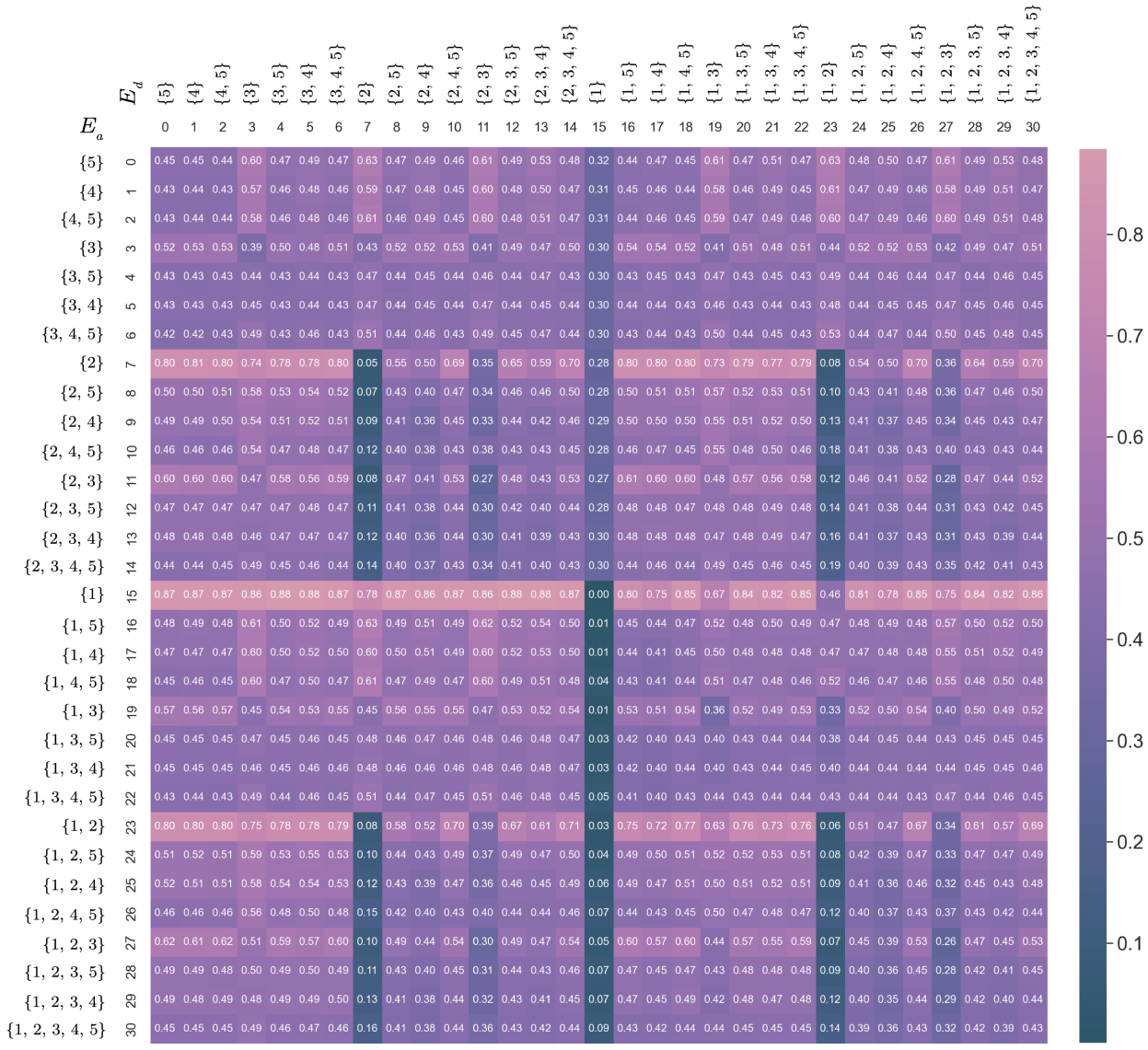


Figure A4. The defender's approximated payoff matrix of **VGG-16** on **SVHN** dataset. The values in the matrix are estimated with a 5-batch evaluation under PGD-20 attack.

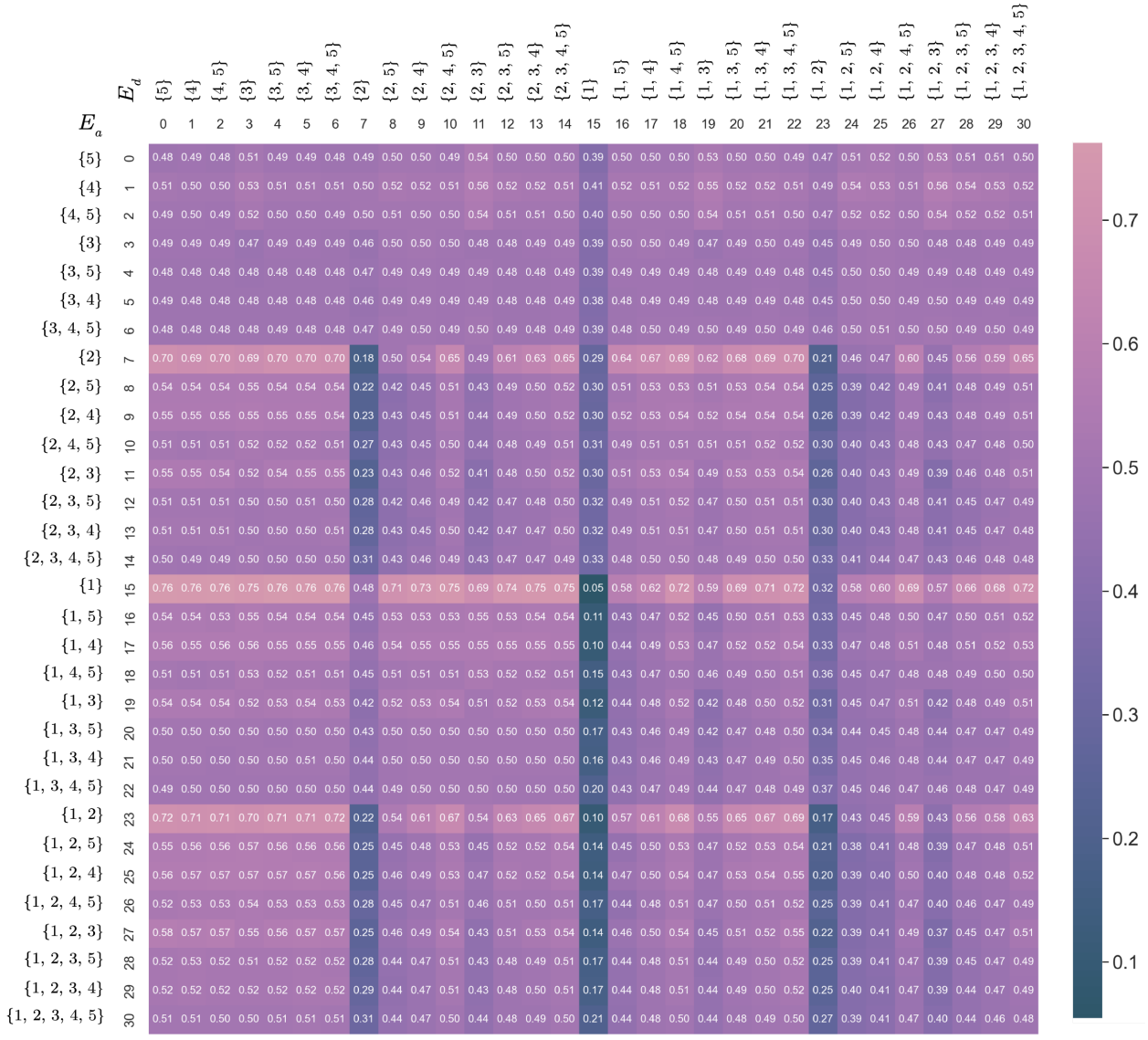


Figure A5. The defender's approximated payoff matrix of **VGG-16** on **CIFAR-10** dataset. The values in the matrix are estimated with a 5-batch evaluation under PGD-20 attack.