Reliable Decision from Multiple Subtasks through Threshold Optimization: Content Moderation in the Wild

Donghyun Son†
ryan.s@hpcnt.com
Hyperconnect
Seoul, South Korea

Byounggyu Lew†
korts@hpcnt.com
Hyperconnect
Seoul, South Korea

Kwanghee Choi†
kwanghee.choi@hpcnt.com
Hyperconnect
Seoul, South Korea

Yongsu Baek
hunter@hpcnt.com
Hyperconnect
Seoul, South Korea

Seungwoo Choi
seungwoo.choi@hpcnt.com
Hyperconnect
Seoul, South Korea

Beomjun Shin
beomjun.shin@match.com
Match Group
Dallas, Texas, USA

Sungjoo Ha
sungjoo.ha@hpcnt.com
Hyperconnect
Seoul, South Korea

Buru Chang*
buru.chang@hpcnt.com
Hyperconnect
Seoul, South Korea

ABSTRACT
Social media platforms struggle to protect users from harmful content through content moderation. These platforms have recently leveraged machine learning models to cope with the vast amount of user-generated content daily. Since moderation policies vary depending on countries and types of products, it is common to train and deploy the models per policy. However, this approach is highly inefficient, especially when the policies change, requiring dataset re-labeling and model re-training on the shifted data distribution. To alleviate this cost inefficiency, social media platforms often employ third-party content moderation services that provide prediction scores of multiple subtasks, such as predicting the existence of underage personnel, rude gestures, or weapons, instead of directly providing final moderation decisions. However, making a reliable automated moderation decision from the prediction scores of the multiple subtasks for a specific target policy has not been widely explored yet. In this study, we formulate real-world scenarios of content moderation and introduce a simple yet effective threshold optimization method that searches the optimal thresholds of the multiple subtasks to make a reliable moderation decision in a cost-effective way. Extensive experiments demonstrate that our approach shows better performance in content moderation compared to existing threshold optimization methods and heuristics.

CCS CONCEPTS
• Computing methodologies → Machine learning approaches;
• Social and professional topics → Censorship.

1 INTRODUCTION
Users of social media platforms that provide user-generated content are always at risk of being exposed to harmful content. To protect the users, most social media platforms operate content moderation systems with human moderators to classify whether a given content is problematic or not [13]. However, human decisions are often noisy [47], and a vast amount of content are generated daily, so it is challenging to handle content only by manual labor. Furthermore, caring for moderators’ mental health while handling harmful content is becoming an important issue [25].

To alleviate the above issues, many social media platforms (e.g., Facebook and Instagram) have utilized machine learning (ML) models to automatically review a large quantity of content without human moderators. However, users can still be exposed to harmful content due to the ML model’s imperfection. This risk imposes a substantial burden on social media platforms under tighter government regulations on inappropriate content [13]. To mitigate this, these platforms integrate the capabilities of both humans and machines: trust the decision of ML models only when their confidence is sufficiently high, while the human moderators handle the rest. Hence, the ML model must make a reliable moderation decision,
at least on par with its human counterparts, to reduce the total volume of content to be handled by the moderators.

To improve the quality of automated decisions, the ML model performance should increase, thus requiring a large-scale dataset. However, it is challenging because the same content may be annotated differently depending on moderation policies varying with country or product. For this reason, the social media platforms need to construct and manage datasets for each moderation policy separately to train and deploy an individual model for the corresponding policy. Furthermore, this approach gradually increases the labeling cost as the moderation policies change due to ever-changing circumstances such as new legislation or societal demand. The existing dataset should be re-labeled under the changing policies, and the pretrained model should also be re-trained on the shifted data distribution [44].

Some social media platforms often employ third-party content moderation services (e.g., Hive moderation1 and Azure content moderator2) to avoid aforementioned issues. As shown in Figure 1, these services provide prediction scores of the multiple subtasks, such as predicting the existence of weapons and rude gestures in a given content, instead of directly providing moderation decisions tailored for each specific service. Customers (social media platforms) make decisions from the provided prediction scores by utilizing or ignoring provided the scores, i.e., defining the decision function (refer to Section 2.3 for the definition). Customers can easily cope with the changing policies by modifying the decision function. To effectively support the customers, moderation services have to define each subtask to be granular, exhaustively covering the potential moderation policies that customers might take.

Although this multiple subtask approach relieves the customers’ burden to operate their own ML models and maintain private datasets, it makes utilizing the prediction scores more complicated. Customers have to first apply thresholding to the prediction scores of the multiple subtasks, instead of directly providing moderation decisions. However, since prediction scores of ML models are often not calibrated [15], it is difficult to determine the optimal thresholds for each subtask that maximize the target metrics (e.g., recall at precision). For example, a threshold of 0.42 could be a high enough threshold for subtask A but not for subtask B. Hence, when we use the same threshold for all the subtask predictions, performance is suboptimal, failing to have effective yet reliable automated decisions.

In this paper, we claim that the sophisticated decision function with the optimal thresholds for prediction scores of multiple subtasks can further improve moderation performance. To do this, we first formulate real-world content moderation scenarios and summarize the concept of the multiple subtask approach. We then propose a simple yet effective threshold optimization method that decides the optimal thresholds of multiple subtasks within a few seconds to maximize the number of automated decisions while maintaining their reliability (i.e., recall at precision). Through experiments on synthetic and real-world moderation datasets, we show that our proposed method outperforms the baselines, including heuristics designed by moderation experts.

Contributions. (1) We introduce real-world content moderation scenarios through the multiple subtask approach to cope with changing moderation policies. (2) We propose an efficient threshold optimization method to make moderation decisions more reliable in the multiple subtask approach where thresholds are found within a few seconds. (3) Our proposed method outperforms existing baselines in both synthetic and real-world moderation datasets.

## 2 MULTIPLE SUBTASK APPROACH

### 2.1 Content Moderation

Content moderation $f$ is commonly defined as classifying whether a given content $x$ is harmless (class 0) or harmful (class 1) under a target policy $p$:

$$f_p(x) = \begin{cases} 
1, & \text{if } x \text{ is harmful}, \\
0, & \text{otherwise.}
\end{cases} \quad (1)$$

Although the performance of ML models has been dramatically improved, considering the risk of missing harmful content, the performance is yet to be perfect to automatically moderate all the content based on the ML models. Therefore, in the real-world, content moderation is designed with the collaboration of the ML models and human moderators to reduce the risk of missing harmful

---

1https://docs.thehive.ai/
2https://azure.microsoft.com/en-us/services/cognitive-services/content-moderator/
As described in previous sections, ML models $m_p$ are often trained and deployed per policy $p$. However, this approach becomes impractical when handling multiple changing policies due to service-specific circumstances, requiring data re-labeling and model retraining. To address this issue, some social media platforms or third-party content moderation services take the multiple subtask approach instead of operating policy-dependent ML models, as shown in Figure 3.

The multiple subtask approach starts with defining $n$ subtasks that are granular enough to cover all the moderation scenarios. For example, one can subdivide hate speech into ethical, racial, and sexual violence words for text content moderation [7]. Since the subtasks focus on detecting highly specified information in the given content rather than policy-dependent moderation decisions, it allows us to continuously accumulate data for each subtask even if policies change. It naturally leads to performance improvement of the ML model targeting a specific subtask.

Let $S = \{s_1, s_2, \cdots, s_n\}$ denote a set of subtasks where each subtask $s_i$ is defined as a binary classification for a given content $x$:

$$s_i(x) = \begin{cases} 1, & \text{if } m_{s_i}(x) > \tau_{s_i}, \\ 0, & \text{otherwise}, \end{cases}$$

where $m_{s_i}(x)$ is the subtask-specific prediction of the ML model $m_{s_i}$ and $\tau_{s_i}$ is the threshold for the subtask. The output of the multiple subtask models is denoted as $O_S(x) = \{s_1(x), s_2(x), \cdots, s_n(x)\} = \{0, 1\}^n$. Either $n$ subtask-specific models [19] or a single multi-task model [42] can be used to implement the above.

### 2.3 Decision Function

Using the multiple subtask output $O_S(x)$, the final moderation decision $f_p(x)$ for the given content $x$ is made with a decision function designed for the policy $p$. The decision function is manually designed as logical operations applied to the output $O_S(x)$ to fulfill the target policy $p$.

Let $S_p = \{s_1^p, s_2^p, \cdots, s_n^p\}$ denote a set of the subtasks $s^p \in S$ which are ground of moderation decision under the target policy $p$. Using the output $O_{S_p}(x) = \{s_1^p(x), s_2^p(x), \cdots\}$, the final moderation decision $f_p(x)$ is made with the decision function $d_p(\cdot) \rightarrow \{0, 1\}$ as follows:

$$f_p(x) = d_p(O_{S_p}(x)) = d_p\left(\left\{ (m_{s^p}(x) > \tau_{s^p}) \right\}_{s^p \in S_p} \right).$$

For a better understanding, let’s assume the situation of content moderation: social media platform for children. The platform should maintain a policy to remove violent images harmful to children. Then, the decision function $d_p$ is defined with boolean operations
AND or OR on the three subtasks, existence of kids $s_{\text{kids}}$, weapons $s_{\text{weapon}}$, and physical violence $s_{\text{violence}}$, as follows.

$$d_p(O_{sp}(x)) = \left( s_{\text{kids}}(x) \text{ AND } s_{\text{weapon}}(x) \text{ OR } s_{\text{violence}}(x) \right).$$  \hspace{1cm} (6)

### 3 Threshold Optimization

As described in the use cases of ML models in content moderation (Figure 2), ML model predictions should be reliable enough to skip manual reviews. When ML models make wrong moderation decisions, we incorrectly restrict harmless content (in Use case 1) or expose harmful content to users (in Use case 2). Hence, it is crucial to make ML model-based moderation decisions more reliable.

What makes moderation decisions reliable? It is known that higher model prediction scores are more reliable, i.e., show higher precision [18]. From this, one may set human precision as the lower bound for automated moderation decisions so that the model is at least not worse than its human counterparts. On the other hand, increasing the recall is also important to lower the moderation cost. As precision and recall have a trade-off relationship, we target to maximize recall at high precision to reduce the moderation cost while preserving the reliability of automated decisions.

There are two possible approaches to improve the final moderation decision in Equation 5. One way is to improve the quality of model predictions $m_p(x)$, which can be challenging in many cases. Another is to optimize the thresholds $r$ for each subtask, balancing them to obtain the optimal decision. We focus on the latter, a much cheaper way to solve the problem than the former.

#### 3.1 Problem Definition

Let $p$ be the target policy and $T_p = \{ t_1^p, t_2^p, \cdots \}$ be the thresholds for the multiple subtasks $S_p = \{ s_1^p, s_2^p, \cdots \}$. We formulate the problem of interest as follows:

$$\begin{align*}
&\text{maximize recall} (T_p|D_p, d_p) \\
&\text{subject to precision} (T_p|D_p, d_p) \geq \text{precision}_t.
\end{align*}$$  \hspace{1cm} (7)

where $D_p$ is a dataset collected for the policy $p$, where each data sample consists of the prediction scores $Q_i = \{ q_{ij}, q_{ij}, \cdots \}$ for the given content $x_i$ and the ground truth decision $y_i^p$ of content moderation. For simplicity, we refer $m_{i,p}(x_i)$ as $q_{ij}$ in the remainder of this paper. recall$(T_p|D_p, d_p)$ and precision$(T_p|D_p, d_p)$ are calculated on the dataset $D_p$ for the given thresholds $T_p$ and the decision function $d_p$ designed for the target policy $p$. Moderation providers target precision, to ensure the reliability of ML-based content moderation.

#### 3.2 Proposed Method

We introduce our threshold optimization method called TruSThresh (Truncated Surrogate gradient for Threshold optimization). TruSThresh involves 1) surrogate gradient learning (SGL) [34] with width truncation to efficiently tune the thresholds, and 2) a penalty method to meet the target precision.

##### 3.2.1 Score Normalization

In real-world scenarios, prediction score distributions are often skewed as each subtask facilitates different data distribution. This skewness makes gradient-based threshold optimization much harder because gradients generated

by the prediction scores would also be skewed. Thus, we normalize the prediction scores $q_{ij}$ for each subtask $s_i$ as follows:

$$\hat{q}_{ij} = \frac{\text{rank}(q_{ij}, \{ q_{ij}, q_{ij}, \cdots , q_{ij} | D_p \})}{|D_p|},$$  \hspace{1cm} (8)

where $|D_p|$ is the number of samples in $D_p$ and rank function $\text{rank}(x, X)$ returns the rank of $x$ in the set $X$ sorted in ascending order. The rank of $q_{ij}$ is divided by $|D_p|$ so that the normalized score $\hat{q}_{ij}$ lies between 0 and 1. Through normalization, we make our optimization strategy more robust by making it agnostic to the prediction score distributions.

We modify Equation 4 with the normalized prediction scores $\tilde{S}_p = \{ \tilde{s}_1^p, \tilde{s}_2^p, \cdots \}$ parameterized by $\tilde{T}_p = \{ \tilde{t}_1^p, \tilde{t}_2^p, \cdots \}$ as follows:

$$\tilde{s}_i(x_j) = \begin{cases} 
1, & \text{if } \hat{q}_{ij} > \tilde{t}_i^p, \\
0, & \text{otherwise},
\end{cases}$$  \hspace{1cm} (9)

where $\tilde{t}_i^p$ is a threshold for the normalized subtask $\tilde{s}_i^p$. Since the normalization does not affect the order of scores, we can easily convert $\tilde{T}_p$ back to $T_p$ using linear interpolation.

##### 3.2.2 Surrogate Gradient Learning with Width Truncation

Similar to Pellegrini and Masquelier [39], our threshold optimization procedure is mainly based on SGL that learns parameters through back-propagation. The backward pass of SGL is computed using the surrogate gradient, while the forward pass is computed using the step function. Pellegrini and Masquelier [39] use the derivative of the sigmoid function as a surrogate gradient while using the Heaviside step function (HSF) in the forward pass. In this work, we also use HSF for the forward pass, but for the backward pass, we approximate the step function using the sine function.

**Forward Pass.** In the forward pass, we first make binary prediction $\tilde{y}^p$ with $\tilde{T}_p$ to compute precision, recall, and loss. The final binary prediction $\tilde{y}^p$ for $f_p$ is as follows:

$$\tilde{y}^p = \tilde{d}_p \left( \{ \tilde{z}_1^p (x), \tilde{z}_2^p (x), \cdots \} \right) = \tilde{d}_p \left( \{ \text{HSF}(\tilde{q}_0^p - \tilde{t}_1^p), \text{HSF}(\tilde{q}_1^p - \tilde{t}_2^p), \cdots \} \right).$$  \hspace{1cm} (10)

We design the numerical version of decision function $\tilde{d}_p$ based on the logical operations of the decision function $d_p$ as follows: Given

![Figure 4: Surrogate Gradient Learning with Width Truncation](image-url)
two inputs $A, B \in \{0, 1\}$, $A$ AND $B$, $A$ OR $B$, and NOT $A$ are substituted with $A \cdot B$, $1 - (1 - A) \cdot (1 - B)$, and $1 - A$, respectively.

Using the predictions $\hat{Y} = \{\hat{y}_0, \hat{y}_1, \ldots\}$ and the ground truth $Y = \{y_0, y_1, \ldots\}$, we calculate precision and recall of our predictions. We further compute the loss $L$ to optimize the parameters $\hat{\gamma}_p$, which will be explained in Subsection 3.2.3.

**Backward Pass.** As all the operations except HSF are differentiable, we define a surrogate gradient $\Theta'$ for $z_{ij} = \hat{q}_{ij} - q_i^*$ as follows:

$$
\Theta'(z_{ij}) = \begin{cases} 
0, & \text{if } |z_{ij}| > w_i, \\
\frac{\partial}{\partial z_{ij}} \left( \frac{1}{2} \sin \left( \frac{\pi z_{ij}}{2w_i} \right) + \frac{1}{2} \right), & \text{if } -w_i \leq z_{ij} \leq w_i, \\
1, & \text{otherwise}. 
\end{cases}
$$

(11)

Note that defining the surrogate gradient as the above is same as approximating the step function HSF as follows:

$$
\text{HSF}(z_{ij}) = \begin{cases} 
0, & \text{if } z_{ij} < -w_i, \\
\frac{1}{2} \sin \left( \frac{\pi z_{ij}}{2w_i} \right) + \frac{1}{2}, & \text{if } -w_i \leq z_{ij} \leq w_i, \\
1, & \text{otherwise}. 
\end{cases}
$$

(12)

The above formulation is motivated from the idea that the threshold should be affected by samples near the threshold. The width $w_i$ determines the amount of samples to be truncated. We parameterize the width $w_i = \sigma(\omega_i)$ so that it lies between 0 and 1. We use the sine function to make the scores closer to the threshold to have a larger effect during optimization.

Given the loss $L$, the gradients with respect to the thresholds are computed using the surrogate gradient as follows:

$$
\frac{\partial L}{\partial \gamma_i^p} = \sum_j \frac{\partial L}{\partial \hat{y}_i^p} \frac{\partial \hat{y}_i^p}{\partial \gamma_i^p} = \sum_j \frac{\partial L}{\partial \hat{y}_i^p} \frac{\partial \hat{y}_i^p}{\partial \text{HSF}(z_{ij})} \frac{\partial \text{HSF}(z_{ij})}{\partial \gamma_i^p} \Theta'(z_{ij})
$$

(13)

While updating the thresholds, we also update the width $w_i$ using a surrogate gradient $\Theta'(w_i)$:

$$
\Theta'(w_i) = \begin{cases} 
0, & \text{if } |z_{ij}| > w_i, \\
\frac{\partial}{\partial w_i} \left( \frac{1}{2} \sin \left( \frac{\pi z_{ij}}{2w_i} \right) + \frac{1}{2} \right), & \text{if } -w_i \leq z_{ij} \leq w_i, \\
1, & \text{otherwise}. 
\end{cases}
$$

(14)

Similar to Equation 14, the final gradients with respect to the widths are computed as follows:

$$
\frac{\partial L}{\partial w_i} \approx \sum_j \frac{\partial L}{\partial \hat{y}_i^p} \frac{\partial \hat{y}_i^p}{\partial \text{HSF}(z_{ij})} \Theta'(w_i)
$$

(15)

We found that making the widths learnable stabilizes training by dynamically adjusting the window size depending on each subtask.

### 3.2.3 Penalty method

To solve the maximization problem (maximizing recall) with constraints (target precision), we design a relaxed loss function $L$ as follows:

$$
L = -\text{recall} + \alpha \max(\text{precision}_r - \text{precision}, 0).
$$

(16)

max(\text{precision}_r - \text{precision}, 0) is a penalty term to ensure that the precision meets the target precision where $\alpha$ controls the strictness of the constraint. In our experiments, we set $\alpha \gg 1$.

## 4 EXPERIMENTS

### 4.1 Experiments on Moderation Use Cases

#### 4.1.1 Settings

We show the performance of various threshold optimization methods on the two moderation use cases (Figure 2). In the first case, the ML model is used to filter out harmful content automatically. The second case covers where the model is utilized to skip harmless content automatically. We compare our method with other baselines in real-world moderation scenarios above. Both scenarios utilize $n$ subtasks to classify whether inappropriate elements are included in the given content (class 1) or not (class 0). Then, we can define decision functions for the two cases as follows:

**Use case 1.**

$$
d(O_1(x)) = \left( s_1(x) \text{ OR } s_2(x) \text{ OR } \cdots \text{ OR } s_n(x) \right)
$$

(17)

**Use case 2.**

$$
d(O_2(x)) = \left( \text{NOT } s_1(x) \text{ AND NOT } s_2(x) \text{ AND } \cdots \text{ AND NOT } s_n(x) \right)
$$

(18)

As we concentrate on the case where we guarantee a reliable moderation decision, i.e., exceed the given target precision, we ignore the case where the precision from the optimized threshold does not surpass the target precision.$^3$

#### 4.1.2 Dataset

We design the experiments to mimic the real-world moderation scenarios by utilizing UnSmile [24], a Korean hate speech dataset. UnSmile contains 15,005/3,737 training/test samples of $n = 9$ hate speech subtasks. We use pretrained model released by the authors to obtain the prediction scores of the validation set and construct the policy-dependent dataset $D_p$.

#### 4.1.3 Baselines

For our method, TrueSThresh, we set the hyperparameters $\tau = 0.5$, $w = 0.1$, and learning rate 0.01 to optimize for 1000 iterations. We compare our method with the following baselines:

**defThresh** [39] uses a single value $\tau$ for all the thresholds, where we use $\tau = 0.5$ in the experiment. greedyThresh calculates recall at precision for each subtask on evenly spaced threshold range between 0 and 1 to iteratively select the best thresholds for each subtask. SGLThresh [39], similar to ours, also employs surrogate gradient of HSF to optimize the thresholds by the gradient descent. It uses $\sigma(z)$ to calculate the surrogate gradient of the step function $I(z > \tau_i)$ where $\tau_i$ is learnable. We set $\tau = 0.3$, $\sigma = 50$, and learning rate 0.001 to optimize for 4000 iterations. Bespoke represents $\hat{f}_p(x)$ in Equation 2 and 3, i.e., a model trained specifically for the target policy, serving as a strong baseline.

#### 4.1.4 Results

Table 1 demonstrates the superior performance of our method compared to the state-of-the-art. Especially, our method shows better performance compared to Bespoke in the second use case, even though it only took a few seconds to optimize for the thresholds. In contrast, defThresh often fails to reach the target precision, and greedyThresh shows limited recall.

---

$^3$The implementation of experiments is publicly released at https://github.com/hyperconnect/trustthresh.
Table 1: Experiments on two moderation use cases (Figure 2). “-” is the case where it fails to meet the target precision. Highest recall for each target precision is highlighted in bold.

| Target Precision | Method            | Recall in Use case 1 | Recall in Use case 2 |
|------------------|-------------------|----------------------|----------------------|
| 0.9              | Bespoke (α = 128) | 0.9069               | 0.2579               |
|                  | delThresh (r = 0.5) | 0.9068               | -                    |
|                  | greedyThresh      | 0.6734               | -                    |
|                  | SGLThresh (α = 32)| 0.9425               | 0.2246               |
|                  | SGLThresh (α = 128)| 0.9422               | 0.2171               |
|                  | TruSThresh (α = 8)| 0.9413               | 0.3080               |
|                  | TruSThresh (α = 32)| 0.9411               | 0.2631               |
|                  | TruSThresh (α = 128)| 0.9372               | 0.2599               |

| 0.95             | Bespoke (α = 128) | 0.8694               | 0.1679               |
|                  | delThresh (r = 0.5) | 0.4882               | -                    |
|                  | greedyThresh      | 0.8451               | -                    |
|                  | SGLThresh (α = 32)| 0.8455               | 0.0246               |
|                  | SGLThresh (α = 128)| 0.8440               | 0.0631               |
|                  | TruSThresh (α = 8)| 0.8512               | 0.1668               |
|                  | TruSThresh (α = 32)| 0.8490               | 0.1679               |
|                  | TruSThresh (α = 128)| 0.8440               | 0.1412               |

| 0.975            | Bespoke (α = 128) | 0.7855               | 0.0235               |
|                  | delThresh (r = 0.5) | 0.3216               | -                    |
|                  | greedyThresh      | 0.7145               | 0.0096               |
|                  | SGLThresh (α = 32)| 0.7077               | 0.0481               |
|                  | SGLThresh (α = 128)| 0.7277               | 0.0995               |

SGLThresh successfully reaches high precision in the first case but shows limited performance with higher precisions in the second case. We suspect the second case is a more challenging scenario, as it is composed of AND operations, significantly decreasing the number of positive samples. Even though increasing the penalty weight α seems beneficial for the second case since it helps reach the target precision by being more strict on the precision constraint, SGLThresh yields a lower recall compared to ours.

We also observe that using larger α degrades the recall for both TruSThresh and SGLThresh although they meet the target precision. In practice, one can increase α until it satisfies the precision constraint, where further increases will negatively impact the recall.

4.2 Experiments on Real-World Content Moderation

4.2.1 Settings. To show the effectiveness of TruSThresh in the wild, we conduct further experiments on the real-world moderation data. We collect 23,596 moderation samples on the live social discovery platform Azar, which has been downloaded over 500 million times. The samples are annotated by professional human moderators. We compare TruSThresh with SGLThresh and heuristic thresholds set by domain experts. These methods search the optimal thresholds of ten subtasks for a decision function designed by the domain experts.

We also observe that using larger α degrades the recall for both TruSThresh and SGLThresh although they meet the target precision. In practice, one can increase α until it satisfies the precision constraint, where further increases will negatively impact the recall.

4.2.2 Results. Figure 5 shows the experimental results on real-world content moderation data. TruSThresh achieves the best recall over all the target precisions compared to the baselines. Especially, at precision 0.985, TruSThresh relatively improves recall by 5.6% and 6.5% compared to SGLThresh and heuristic thresholds, respectively. We emphasize that TruSThresh reduces a considerable amount of annotation cost by only tuning the thresholds. Moreover, unlike TruSThresh, SGLThresh fails to meet higher target precisions (> 0.985). These results verify that TruSThresh is highly effective in real-world content moderation systems.

4.3 Experiments on Different Target Metric

4.3.1 Settings. We follow the settings of [39], where it optimizes the thresholds to maximize micro-averaged F1 score. The score is defined in the multi-label classification setting as follows:

\[
\text{Micro-averaged F1} = 2 \frac{\sum_{c,n} y^c_n \hat{y}^c_n}{\sum_{c,n} y^c_n + \sum_{c,n} \hat{y}^c_n}.
\]

4.3.2 Dataset. Similar to [39], we use DCASE17 [33] and DCASE19 [43], which is the multi-labeled sound event detection dataset. We extend the experiment to different modalities by adding EurLex [3] and CelebA [28], which is the multi-label text and image dataset, respectively. We target the validation set if there

based on the application’s moderation policy. Note that we cannot report confidential information such as the ratio between true/false classes, details of each subtask, and the exact recall value.

4.4.2 Results. Figure 5 shows the experimental results on real-world moderation data. TruSThresh achieves the best recall over all the target precisions compared to the baselines. Especially, at precision 0.985, TruSThresh relatively improves recall by 5.6% and 6.5% compared to SGLThresh and heuristic thresholds, respectively. We emphasize that TruSThresh reduces a considerable amount of annotation cost by only tuning the thresholds. Moreover, unlike TruSThresh, SGLThresh fails to meet higher target precisions (> 0.985). These results verify that TruSThresh is highly effective in real-world content moderation systems.

4.3 Experiments on Different Target Metric

4.3.1 Settings. We follow the settings of [39], where it optimizes the thresholds to maximize micro-averaged F1 score. The score is defined in the multi-label classification setting as follows:

\[
\text{Micro-averaged F1} = 2 \frac{\sum_{c,n} y^c_n \hat{y}^c_n}{\sum_{c,n} y^c_n + \sum_{c,n} \hat{y}^c_n}.
\]

y^c_n and \( \hat{y}^c_n \) are the ground truth and the prediction, respectively, for class \( c \) and instance \( n \).

We compare the methods using the following task:

\[
\maximize \text{ Micro-averaged F1}(T(D)),
\]

where \( T \) is a set of class-wise thresholds and \( D \) is the target multi-label classification dataset.

4.3.2 Dataset. Similar to [39], we use DCASE17 [33] and DCASE19 [43], which is the multi-labeled sound event detection dataset. We extend the experiment to different modalities by adding EurLex [3] and CelebA [28], which is the multi-label text and image dataset, respectively. We target the validation set if there
Table 3: Experiments on Micro-averaged F1 Score.

| Method      | DCASE17 | DCASE19 | CelebA | EurLex |
|-------------|---------|---------|--------|--------|
| defThresh   | 0.566   | 0.706   | 0.772  | 0.564  |
| greedyThresh| 0.636   | 0.728   | 0.771  | 0.593  |
| SGLThresh   | 0.627   | 0.731   | 0.783  | 0.641  |
| TruSThresh  | 0.641   | 0.732   | 0.783  | 0.639  |

Table 4: Ablation study on Score Normalization and Width Tuning. Experiments are conducted on Use case 2 with \( \alpha = 32 \).

| Target Precision | Score Normalization | Width Tuning | Precision | Recall |
|------------------|---------------------|--------------|-----------|--------|
| 0.9              | ✗                   | ✗            | 0.9286    | 0.1807 |
| 0.95             | ✗                   | ✓            | 0.9272    | 0.1497 |
| 0.95             | ✓                   | ✓            | 0.9046    | 0.2535 |
| 0.95             | ✓                   | ✓            | 0.9011    | 0.2631 |
| 0.95             | ✓                   | ✓            | 0.9737    | 0.0396 |
| 0.95             | ✓                   | ✓            | 0.9697    | 0.0342 |
| 0.95             | ✓                   | ✓            | 0.9506    | 0.1647 |
| 0.95             | ✓                   | ✓            | 0.9518    | 0.1679 |
| 0.95             | ✓                   | ✓            | 1.0000    | 0.0000 |
| 0.95             | ✓                   | ✓            | 1.0000    | 0.0000 |
| 0.95             | ✓                   | ✓            | 0.9810    | 0.1102 |
| 0.95             | ✓                   | ✓            | 0.9767    | 0.1348 |

Table 5: Comparing SGLThresh with TruSThresh. Experiments are conducted on Use case 2 with \( \alpha = 32 \). For SGLThresh+SN, we apply score normalization to SGLThresh.

| Target Precision | Method     | Precision | Recall |
|------------------|------------|-----------|--------|
| 0.9              | SGLThresh  | 0.9013    | 0.2246 |
|                  | SGLThresh+SN| 0.9026    | 0.2578 |
|                  | TruSThresh  | 0.9011    | 0.2631 |
| 0.95             | SGLThresh  | 0.9583    | 0.0246 |
|                  | SGLThresh+SN| 0.9517    | 0.1476 |
|                  | TruSThresh  | 0.9518    | 0.1679 |
| 0.95             | SGLThresh  | 1.0000    | 0.0096 |
|                  | SGLThresh+SN| 0.9828    | 0.1219 |
|                  | TruSThresh  | 0.9767    | 0.1348 |

5 ANALYSIS

5.1 Ablation Study

As mentioned in Subsection 3.2, our method contains two key components: Score Normalization (SN) and Width Tuning (WT). In Table 4, we observe the performance when we remove each component. We use the strictness of \( \alpha = 32 \) for all the experiments. We verify the effectiveness of both components, where SN greatly helps precisely target the precision. As a result, recall increases while satisfying the precision constraint.

5.2 Comparison with SGLThresh

Applying Score Normalization to SGLThresh. To the best of our knowledge, SGLThresh [39] is the first work that applied surrogate gradient learning for threshold optimization. To show the capability of our SN component, we attach the component to SGLThresh, comparing the downstream performance. Original SGLThresh does not contain any score normalization involved. We use the same settings of Subsection 5.1. Table 5 demonstrates that our SN component is effective, similar to Subsection 5.1. SN helps increase the recall by setting the precision to be closer to the target precision.

5.3 Optimization Analysis

The above subsection shows that the performance of SGLThresh is inferior to that of TruSThresh even with score normalization. To further investigate the difference between the two methods, we analyze how learnable parameters are optimized during training. The main difference comes from dissimilar approximations of HSF. Given the thresholded prediction \( z = q - \tau \), our method uses the sinusoidal function with a learnable width size, where SGLThresh uses \( \text{sigmoid}(\sigma z) \) with learnable \( \sigma \). Both methods control the spread that depends on each subtask via classwise \( w \) and \( \sigma \), respectively.

To observe whether each parameter is sufficiently being controlled to match the characteristics of each subtask, we plot how subtask-specific parameters are changing, i.e., plot \( \sigma \) and \( w \) where \( T \) is the number of optimization steps. We use the same settings from Subsection 4.1, where each method has to optimize nine parameters per subtask. We set the initial \( \sigma_{\text{def}} = (10, 25, 50) \) and width \( w_{\text{def}} = 0.1 \). We also test smaller \( \sigma \) than the original implementation of SGLThresh, which uses \( \sigma = 50 \), to check whether...
The above equation yields a small maximum value around $\sigma = 50$. We suspect that the gradient of SGLThresh for each $\sigma$ vanishes, while our approximated HSF avoids this problem.

Finally, we compare the optimization speed in Figure 6 (d). Even though the speed of SGLThresh significantly improves after applying score normalization, TruSTHresh is the quickest to converge, having the computational upper hand compared to SGLThresh. We suspect that the design of subtask-specific width and the unimportant samples’ gradients being truncated makes our method quickly optimize to the optimal thresholds with less number of steps.

### 6 RELATED WORK

#### 6.1 Machine Learning for Content Moderation

ML models have been widely adopted for content moderation in the last decade. Since it is very difficult to obtain data on content moderation due to the sensitivity of harmful content, studies on content moderation are mainly conducted in industry rather than academia. In the industry, research on collaboration between ML models and human moderators is predominant [14, 31, 32], rather than directly revealing the ML models operating on social media platforms. Studies conducted in academia mainly focus on text content that is easily available online. Hate speech detection is one of the popular research topics on content moderation. Using benchmark datasets obtained from Twitter [6, 12, 45], Yahoo! [36], and Reddit [40], hate speech detection models [1, 8] have been proposed based on the state-of-the-art architecture such as convolutional neural networks [38], recurrent neural networks [11], and Transformer [10]. Several studies [4, 5] have proposed spoiler detection models to protect users from spoilers that ruin the pleasures of the users for the creative works. Recently, a detection model [20] tracking harmful content that sells illicit drugs on Instagram has been proposed based on deep multi-modal multi-label learning.

#### 6.2 Threshold Optimization

Several research areas are loosely related to our study in terms of finding the optimal thresholds that maximize the target metrics. When we formulate the threshold optimization problem as minimizing a real-valued function output, we can consider it as the unconstrained nonlinear optimization problem [35, 37]. Also, the algorithm configuration problem [22] concentrates on the variant of the above situation where yielding the output is expensive. However, calculating the metrics such as the F1 score is a cheap operation compared to the targets of the algorithm configuration problem. Even though our work did not concentrate on improving the model itself, there are recent efforts that directly optimize the model to the black-box metrics. Jiang et al. [23] estimate the gradients of the metrics to utilize, and Huang et al. [21] address the mismatch between the loss to train the model and the metric to evaluate, where they adjust the loss to follow the metric. Furthermore, some concentrate on ranking-based metrics, such as precision or recall, similar to our work. Eban et al. [9] apply relaxation to the metrics to make it tractable, and Revaud et al. [41] and Henderson and Ferrari [17] utilize mean average precision loss to solve image retrieval or object detection problems. The most similar work is the threshold optimization literature [2, 26, 39]. Kong et al. [26] directly optimize the F1 score via simple gradient descent with heuristically chosen step size. Cances et al. [2] search the thresholds within a coarse range to iteratively reduce the search space. Finally, SGLThresh [39] improve the previous two methods, where our method share the key idea of surrogate gradient learning.

### 7 CONCLUSION

In this study, we describe real-world content moderation scenarios based on the multiple subtask approach to cope with various moderation policies. To make moderation decisions more reliable from multiple subtask predictions of ML models, we propose a threshold optimization method that finds the optimal thresholds for the subtasks. Experiments on synthetic and real-world content moderation datasets show that our proposed method improves recall while preserving high precision, optimizing within seconds without any model parameter updates. We believe our study will aid social media platforms that already operate ML-based content moderation systems or are considering building new systems.
