# An Analysis under a Unified Formulation of Learning Algorithms with Output Constraints

**Mooho Song**
Seoul National University
anmh9161@snu.ac.kr

**Jay-Yoon Lee**
Seoul National University
lee.jayyoon@snu.ac.kr

## Abstract

Neural networks (NN) excel in diverse tasks but can produce nonsensical results due to their exclusive reliance on (input, output) pairs, often conflicting with human knowledge. Injecting human knowledge via output constraints can enhance performance and reduce violations. Despite attempts to compare existing algorithms, no unified categorization of learning algorithms with output constraints exists. Our contributions are: (1) We categorize previous studies using three axes: type of constraint loss (e.g., probabilistic soft logic, REINFORCE), exploration strategy of constraint-violating examples, and integration mechanism for balancing the main task and constraints learning signals. (2) We propose new algorithms inspired by continual-learning for integrating main task and constraint information. (3) We introduce the $H\beta$-score metric to simultaneously evaluate main task performance and constraint violation. Our experiments on NLP tasks (NLI, STE, SRL) show that our projection-based integration mechanism outperforms others. Sampling strategy is crucial for high $H\beta$-scores, with better results as sample numbers increase. Additionally, soft-type constraint loss performs well when combined with sampling strategies. These insights highlight key factors for achieving high $H\beta$-scores and demonstrate the efficacy of our methods.

## 1 Introduction

The majority of neural networks (NN) models "solely" learn from data in the form of (input, output) pairs, and such models can sometimes result in a conflict with human knowledge. Previous work has shown that injecting human knowledge into NN models in the form of reducing relevant constraint violations during training time can improve the model performance as well as reducing constraint violations (Li et al., 2020; Nandwani et al., 2019; Mehta et al., 2018; Rajaby Faghihi et al., 2023; Xu et al., 2018). The relation between constraint and the task itself can be viewed as a relation between the sub-task and the main task. The goal of the main task would be to acquire the most accurate prediction possible, whereas the goal of the subtask is to simply acquire constraint-satisfying output. The focus in injecting constraint is to preserve or improve the main task performance while improving the constraint satisfaction.

Various literature exists on constraint injection during training time (Li et al., 2020; Nandwani et al., 2019; Mehta et al., 2018), where the majority of them formulates the loss function as an addition of loss related to constraint to the existing supervised loss term. Research on how to formulate the loss related to constraint and how much to incorporate in comparison to existing supervised loss is scattered as these approaches vary across different studies and applications.

The first goal of this work is to provide a unified analysis of existing methods from previous studies under a single mathematical formulation. Early efforts to compare different constraint injection methods (Rajaby Faghihi et al., 2023) do exist, however, their focus was on comparing performances of different algorithms, as presented in previous work. On the other hand, our study aims to formalize previous literature from a new unified perspective, to understand key success factors in existing algorithms. For example, while primal-dual algorithm (Nandwani et al., 2019) have shown positive results with the idea of dynamic weight update on constraint-loss, it was only tested under the single loss type of Probabilistic Soft Logic (PSL) (Bröcheler et al., 2010). This makes it unclear whether the positive results were contributions mostly coming from PSL or from the novel dynamic weight update algorithm. As the same weight update mechanism can be applied to different loss types, such as REINFORCE loss, it is worthwhile to investigate mixing and matching different components of injecting constraint under a unified formulation. While numerous studies have focused on injecting constraint during training time, to the best of our knowledge, there has been no research consolidating these studies into a unified mathematical formulation to compare their characteristics component by component.

The second goal of this work is to propose new effective learning algorithms that integrate constraints within the suggested unified formulation. A common approach to learning with constraints involves handling a constraint loss term, $\lambda \times \mathcal{C}$, where $\mathcal{C}$ denotes the constraint loss and $\lambda$ is a fixed scalar representing the weight of $\mathcal{C}$. By adding $\lambda \times \mathcal{C}$ to the pre-existing supervised loss term. Nandwani et al. introduced an algorithm that dynamically controls $\lambda$, starting training with $\lambda$ at 0 and progressively adjusting its value during the learning process. This algorithm is characterized by the gradual increase of the weight $\lambda$, updating it solely based on the degree of constraints. While the work of Nandwani et al. is distinguished from existing methods that use a fixed hyperparameter $\lambda$, there has not been sufficient research for integrating the learning signals form supervised data and constraint information beyond this work.

Is it always necessary to have the value of $\lambda$ monotonically increasing for training? Is there a way to update the value of $\lambda$ considering both supervised learning and constraint injection? Inspired by continual learning methods, this paper proposes a new approach that considers both supervised loss and constraint loss during gradient updates. This approach takes into account the progress of both tasks: supervised learning and constraint injection tasks. It offers a new viewpoint for injecting constraint on simultaneously learning these two tasks. Experiments demonstrate that our new approach achieves the highest-level of performance other learning algorithms in various scenarios.

## 2 Unified formulation of previous work

In this section, we categorize the previous studies on injecting constraints during training time based on three dimensions: type of mathematical expression used for constraint loss (§2.1), exploration strategy of constraint-violating examples (§2.2), and mechanism for integrating losses from the main task and the constraint injection task (§2.3). A common approach in machine learning is to define a loss function and employ optimization algorithms to update model parameters in the direction of minimizing that loss. When the labeled data $\{(x_i, y_i)\}_{i=1}^N$ is given, the goal of typical supervised learning is to solve the following optimization problem:

$$\min_\theta \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(x_i, y_i; \theta), \quad \text{or simply} \quad \min_\theta \mathcal{L}(\theta) \quad (1)$$

, where $\mathcal{L}(x, y; \theta)$ is the standard supervised loss function for the task we are learning.

Most of the existing constraint injection methods, while differing in specific formulations, inject the constraint information by expanding the loss function in a following manner:

$$\mathcal{T}(\theta) = \lambda_1 \mathcal{L}(\theta) + \lambda_2 \cdot \mathcal{C}(\theta) \quad (2)$$

, where $\mathcal{C}(\theta)$ is a loss related to the constraint, and $\lambda_i$'s are fixed weights, We can further generalize the equation (2) as follow:

$$\nabla \mathcal{T}(\theta) = \Lambda^{sup} \cdot \nabla \mathcal{L}(\theta) + \Lambda^{con} \cdot \nabla \mathcal{C}(\theta) \quad (3)$$

, where $\Lambda^{sup}$, $\Lambda^{con}$ are usually scalar matrices. $\mathcal{C}$ reflects the human knowledge the algorithm wants to inject and is typically computed without the true label. To be more precise, for some hard constraints on output labels, $\mathcal{C}(\theta)$ is computed via output $f_\theta(x)$ given some unlabeled input $x$. Injecting more than one hard constraint is also possible by expanding $\Lambda^{con} \cdot \nabla \mathcal{C}(\theta)$ to $\sum_{i=1}^{K} \Lambda_i^{con} \cdot \nabla \mathcal{C}_i(\theta)$ in equation (3) , where $K$ is a number of constraints.

To unify and distinguish different algorithms that learn with constraints, we focus on how $\mathcal{C}(\theta)$ is formulated (§2.1), how constraint-violating examples are explored (§2.2), and how $\Lambda^{sup}$, $\Lambda^{con}$ (in equation (3)) are determined(§2.3).

### 2.1 Type of constraint loss

Type of constraint loss is related to how the violation of constraints can be transformed into the form of a differentiable loss function $\mathcal{C}(\theta)$ in equation (2), which we will refer to as the *constraint loss*. How to convert symbolic constraints into a differentiable loss function can be broadly categorized into two approaches: Probabilistic Soft Logic (PSL) and REINFORCE.

**Probabilistic Soft Logic (PSL)** PSL (Bröcheler et al., 2010) is associated with expressing logic in terms of probabilities, and research utilizing PSL measures the degree of constraint violation in the logic itself, employing it as a loss. Gödel, product, Łukasiewicz logics can be primarily used to soften logic (Minervini and Riedel, 2018; Nandwani et al., 2019; Li et al., 2020), and these examples are listed in table 1. Generally, PSL is not suitable for representing all types of hard constraints, since it must be converted to linear constraints before they can be directly applied (Rajaby Faghihi et al., 2023). Section §4.2 is an example task illustraing the challenges in applying PSL, and the more details are in Appendix §A.2.

**REINFORCE** In contrast, studies employing the REINFORCE (Williams, 1992) evaluate whether (or to what degree) the model's output violates constraints. Constraint injection research during training time using REINFORCE can be further classified into two ways depending how the reward is formulated. A simple method is to assign binary reward (e.g.: $\{1, 0\}$) when the model satisfies or violates the constraints (Ahmed et al., 2022). This simple method with binary reward only considers whether the constraint is satisfied or not. On the other hand, one could make the reward more fine-grained by measuring the degree of constraint violation and assigning real-valued rewards related to it (Mehta et al., 2018). A significant feature of REINFORCE is that the determination of constraint loss relies solely on the rule of assigning rewards based on the presence or absence of constraint violation in sampled examples, regardless of the specific constraint. This differs from PSL in that it does not require intricate implementations for generating constraint loss. However, due to the need for sampling procedures, the computational cost is generally higher than when using PSL (Rajaby Faghihi et al., 2023).

To summarize, PSL and REINFORCE are mainly used approach to generate $\mathcal{C}(\theta)$ in eq.(2) to reduce expected constraint violation with following differences. PSL defines constraint violation as a continuous measure, while REINFORCE relies on the reinforcement learning paradigm to guide the model towards satisfying constraints. Specifically, the REINFORCE method is divided into two types based on the method of setting rewards: binary rewards and real rewards. More specific comparison between types of constraint losses: PSL and REINFORCE is in Appendix §A.1.

## 2.2 Exploration of constraint-violating examples

Let $f_\theta(x)$ represent the output distribution associated with the model $f$ parameterized by $\theta$ given input $x$. The identification of constraint-violating examples from $f(x)$ plays a crucial role in determining constraint loss $\mathcal{C}(\theta)$. Therefore, exploration of constraint-violating examples can significantly impact the effectiveness and efficiency of constraint learning. The possible questions we have are as follows: Would it be better to explore the model's approximate output space? Would it be best to examine the model's best possible effort? Or would it be better to explore by considering all possible probability distributions? Theses are considered to determine the magnitude of the constraint loss $\mathcal{C}$. For example, in REINFORCE with $\{1, 0\}$ reward, the reward will be 0 if we only visit constraint-violating examples.

According to the questions posed above, exploration strategies are divided by three, each explained below: *sampling*, *argmax*, and *exhaustive*.

**Sampling** Sampling strategy involves drawing samples from the forward propagation results of the model $f$ to examine different instances that violate constraints. As demonstrated by (Ahmed et al., 2022), this method commonly employs the REINFORCE algorithm to incorporate constraint violations into the loss function for the identified examples. The sampling strategy can be applied independently to all combinations for our other analysis axes, specifically concerning the type of constraint loss (§2.1) and the integration mechanism of learning signals from main task and constraint (§2.3).

**Argmax (Top-1)** Argmax (Top-1) strategy, constraint violation is assesed by choosing the combination with the highest probability from $f(x)$. Following greedy decoding process such as beam search or Vitrerbi decoding (Mehta et al., 2018), it evaluates constraint violation for the decoded example. Similar to sampling, it evaluates constraint violation for the decoded example, but distinguishes itself by considering the most probable prediction at that moment without multiple samplings. Like the sampling strategy, the argmax strategy can also be applied independently to all combinations under our other axes of anlysis.

**Exhaustive** Exhaustive strategy considers probabilities of all output class and its combinations.

It is prominently employed in research related to PSL (Nandwani et al., 2019; Li et al., 2020). Since there is no sampling involved, it is computationally cost-effective rather than sampling strategy. When considering the type of constraint loss (§2.1), our performance evaluation is exclusively conducted using PSL for the exhaustive strategy, excluding the REINFORCE in the constraint loss, as it would be impossible to consider all possible combinations in REINFORCE. Since the exhaustive strategy can only be applied for constraint loss type of PSL, exhaustive strategy cannot handle all of general type of constraints.

### 2.3 Integration mechanism of learning signals from main task and constraint

This section is related to the integration of main task and the constraint information. We categorize integration mechanisms of prior studies into *static* and *monotone ($\lambda$ ↑)*. Additionally, we introduce three new integration mechanisms based on the linear projection: *projection-sup*, *projection-con*, and *projection-both*, which will be discussed in section §3. We provide detailed explanations of these mechanisms below.

**Static** For constraint loss $\mathcal{C}$, a widely used approach incorporating $\mathcal{C}$ into the existing supervised loss term is to add $\lambda \cdot \mathcal{C}$ to the previous existing loss, where $\lambda$ is a fixed positive real number (Ahmed et al., 2022; Li et al., 2020; Mehta et al., 2018; Minervini and Riedel, 2018). In this approach, the value of $\lambda$ remains unchanged throughout the training process, serving as a constant multiplier that determines the relative influence of $\mathcal{C}$ in comparison to the main task loss $\mathcal{L}$ in eq.(2).

**Monotone ($\lambda$ ↑)** On the other hand, the study by (Nandwani et al., 2019) deviates from this by not using a fixed $\lambda$. Instead, it initiates training with $\lambda$ starting from 0 and progressively adjusting its value during the learning process. This concept emerged from the transformation of the constrained optimization problem into a max-min problem, employing alternative updates. In this method, the value of $\lambda$ steadily grows throughout the training, signifying a progressive emphasis on the constraint loss.

**Projection** Unlike previous methods, projection methods perform gradient updates considering the gradients of two losses: $\mathcal{L}$ and $\mathcal{C}$. For both *static* and *monotone ($\lambda$ ↑)*, $\Lambda$'s are all diagonal matrices in equation (3). However, the projection method results in non-diagonal matrices depending on the gradients of both loss functions. The detailed formulation will be introduced in section §3.

It is important to note that the decision on how to integrate two losses ($\mathcal{L}$, $\mathcal{C}$) is entirely separate from the process of formulating the constraint loss $\mathcal{C}$ (§2.1), and the exploring strategy of constraint-violation examples (§2.2). Therefore, adjusting $\Lambda$'s (or, $\lambda$'s) mentioned in this section can be independently combined with other analysis axes.

## 3 Further exploration on integration of main task and constraint information

In this section, we propose new methods for integrating the losses of main task and constraint injection task. Departing from categorized methods used in previous research, 'static' and 'monotone($\lambda$ ↑)', we introduce three new integration mechanism for the two losses: 'projection-sup', 'projection-con', and 'projection-both'.

**Motivation** Gradient Episodic Memory (GEM) (Lopez-Paz and Ranzato, 2017) model is designed for continual learning for positive backward transfer, aiming to store memories of previous tasks in such a way that the loss does not increase when learning from new data. It introduces constraints to prevent an increase in loss for previous tasks stored in memory when learning from new data and presents a new minimization problem. A-GEM (Chaudhry et al., 2018) is a variant of GEM that is designed for effective memory and computational cost, by storing the averaged episodic memory across the all tasks. Motivated by these works, we propose a new method for integrating losses – $\mathcal{L}(\theta)$ and $\mathcal{C}(\theta)$ – for two tasks. In GEM/A-GEM, whenever new data was deemed to violate positive backward transfer, it applies a projection operation for the gradients to adjust them. We adapt the concept from GEM/A-GEM and utilize it in designing the integration mechanism of main task and constraint information

**Method** Recall that the derivative of loss function with constraint has form of:

$$\nabla\mathcal{T}(\theta) = \Lambda^{sup} \cdot \nabla\mathcal{L}(\theta) + \Lambda^{con} \cdot \nabla\mathcal{C}(\theta)$$

Our approach is rooted in the idea that supervised learning and constraint injection are two distinct tasks, and during their respective updates, we can prevent negatively effecting each other by executing a projection of gradient for each other. The followings are explanations of three new algorithms, and the pseudo-codes are available in Appendix C.

*Projection-sup* applies the projection method to the gradient of the constraint loss (namely, adjust $\Lambda^{con}$) to prevent it from negatively affecting the supervised learning task, while storing the averaged gradient vector of supervised learning task $g_{sup}$ previously used for training. Mathematically, project $\nabla\mathcal{C}(\theta)$ via:

$$\text{Proj}(\nabla\mathcal{C}(\theta)) = \nabla\mathcal{C}(\theta) - \frac{\nabla\mathcal{C}(\theta)\cdot g_{sup}}{g_{sup}\cdot g_{sup}} g_{sup} \quad (4)$$

, whenever $\nabla\mathcal{C}(\theta)\cdot g_{sup} < 0$. Then, the vector $\text{Proj}(\nabla\mathcal{C}(\theta))$ satisfies $\text{Proj}(\nabla\mathcal{C}(\theta))\cdot g_{sup} = 0$. This ensures that $\nabla\mathcal{C}$ is transformed orthogonally to $g_{sup}$, preventing it from providing information that contradicts supervised learning.

Conversely, *projection-con* applies the projection method to the gradient of the supervised loss (namely, adjust $\Lambda^{con}$) to prevent it from negatively affecting the constraint injection task, while storing the averaged gradient vector of constraint injection task $g_{sup}$ previously used for training. Mathematically, project $\nabla\mathcal{L}(\theta)$ via:

$$\text{Proj}(\nabla\mathcal{L}(\theta)) = \nabla\mathcal{L}(\theta) - \frac{\nabla\mathcal{L}(\theta)\cdot g_{con}}{g_{con}\cdot g_{con}} g_{con} \quad (5)$$

, whenever $\nabla\mathcal{L}(\theta)\cdot g_{con} < 0$. Then, the vector $\text{Proj}(\nabla\mathcal{L}(\theta))$ satisfies $\text{Proj}(\nabla\mathcal{L}(\theta))\cdot g_{con} = 0$. This ensures that $\nabla\mathcal{L}$ is transformed orthogonally to $g_{con}$, preventing it from providing information that contradicts constraint injection.

*Projection-both* combines both projection-sup and projection-con, applying projection to both gradients (namely, adjust both $\Lambda^{sup}$ and $\Lambda^{con}$) to ensure that neither task negatively impacts the other. It stores two types of gradients separately by each task used for training before, and apply two projections (4) and (5) together.

## 4 Tasks

In this section, we introduce the tasks for which we conduct experiments: Natural Language Inference (NLI), Synthetic Transduction Example (STE), and Semantic Role Labeling (SRL). Additional details about tasks and implementations are explained in Appendix §D.

### 4.1 Natural Language Inference (NLI)

NLI is a task that involves understanding the logical relationships between pairs of text. Given a premise (P) and a hypothesis (H), the task is to determine whether P entails H, contradicts H, or maintains a neutral relationship with H. There exists constraints such as if P entails H, then H must not contradict P. We used the five constraints listed in (Minervini and Riedel, 2018), as shown in table 4 . The dataset used is SNLI (Bowman et al., 2015).

### 4.2 Synthetic Transduction Example (STE)

We also present an artificial task utilized in (Lee et al., 2019). A sequence transducer $T : \mathcal{L}_S \rightarrow \mathcal{L}_T$ converts the source language $\mathcal{L}_S = (az|bz)^*$ to the target language $\mathcal{L}_T = (za|bbb)^*$, for example, $T(azbzbz) = zabbbbbb$. The constraint imposed involves the relationship between the number of 'b' in the source and the target. Specifically, the count of 'b' in the target must be exactly three times that in the source.

### 4.3 Semantic Role Labeling (SRL)

SRL is a natural language processing task that predicts the semantic roles of each word in a sentence with respect to a given verb or predicate. The method of our work employed for this purpose is BIO tagging.

The Unique Core Roles constraint from (Li et al., 2020) is applied as a constraint, which means that there can be no more than one occurrence of each core argument. For a predicate $u$, if the model predicts the $i$-th word as `B-X`, then other words in the same prediction should not be predicted as `B-X`. This can be expressed as follow.

$$\begin{aligned} &\forall\, u, i \in s, \mathtt{X} \in \mathcal{A}_{core}, \\ &\quad B_{\mathtt{X}}(u,i) \rightarrow \bigwedge_{j\in s, j\neq i} \neg B_{\mathtt{X}}(u,j). \end{aligned} \quad (6)$$

The dataset we used is English Ontonotes v5, with the CoNLL-2012 shared task format (Pradhan et al., 2012).

## 5 Experiments

Our experiment is composed of NLI, STE, and SRL tasks, with accuracy, token accuracy, and F1 score are used as the main task metrics, respectively. Our goal is to first observe the performance trends of algorithms according to our three classification criteria. Then, we will explore combinations that show particularly strong performance.

**Experiment environment** We used RTX 3090 GPU, and Adam optimizer for all of trainings. We conducted training for each case 10 times, and the results are displayed as the mean (in the larger font above) and standard deviation (in the smaller font below) for both the main task metric (denoted by Perf) and constraint violation (denoted by Const.Vio)[1] rate.

**Metric** Comparing the superiority of experimental results considering two different metrics simultaneously is very challenging, especially when there is no occurrence of Pareto-improvement. The *$H\beta$-score* (Harmonic $\beta$ Score) we propose is an indicator that allows for a quick and clear evaluation of experimental outcomes based on two metrics. Assume we have two metrics to consider, and denote the scores for each metric as $m_1$ and $m_2$, respectively. Both metrics are assumed to have values ranging from 0 to 1, with higher values indicating better performance[2]. The $H\beta$-score is similar in form to the F$\beta$-score and is defined as follow:

$$H_\beta(m_1, m_2) = \frac{1+\beta^2}{\frac{1}{m_1} + \frac{\beta^2}{m_2}}.$$

The $H\beta$-score is exactly the same in form as the F$\beta$-score. It is simply an extension of the F$\beta$-score, which uses precision and recall as arguments, to be a score for any two arbitrary metrics. If the magnitude of $\beta$ increases, the evaluation significantly considers the weight of $m_2$. Conversely, as the value of $\beta$ approaches zero, the weight of $m_1$ is significantly considered in the evaluation.

**Experiment results** Table 3 shows the experiment results for all combinations possible in our analysis axes which consist of previous methods and our newly proposed methods. For each task, we present the experimental results based on our three analysis axes proposed in section §2: type of constraint loss (soft, binary, real), exploration strategy of constraint-violating examples (top-1, sampling, exhaustive), and mechanism for integrating the main and the constraint information (static, monotone, proj-sup, proj-con, proj-both).

As the sheer number of experiments is too large to interpret in table 3, we try to examine key factors for best main task performance, constraint violation by dissecting the table 3 from different perspectives.

**Trends per analysis axes** Figure 4 illustrates the top 5 experimental results with the highest $H\beta$-scores for each of the five integration mechanisms described in section §2.3. Among the five integration mechanisms, *projection-con* and *projection-both* consistently demonstrates the best performance across most $\beta$ values. They excel in a wide range of scenarios, from those emphasizing main task metrics (lower $\beta$ values) to those prioritizing constraint injection task performance (higher $\beta$ values). The static and monotone mechanism seldom performs well , they do not always exhibit excellent performance across all tasks.

Figure 5 illustrates the top 5 experimental results with the highest $H\beta$-scores for each of the three types of constraint losses described in section §2.1. Among the three types of losses, whether soft or real type shows consistently better performance depends on the task. Our hypothesis is, as mentioned in appendix A.1, soft and real types of losses can incorporate more fine-grained information into constraint loss compared to binary types of loss.

Figure 6 illustrates the top 5 experimental results with the highest $H\beta$-scores, considering each of the five exploring strategies described in section §2.2. Among the five strategies, one clear observation is that the sampling method consistently demonstrates superior performance across all tasks. Although there are variations, performance tends to improve as the sample size increases. However, the overall performance of the full strategy is not favorable, especially in SRL task. In the full strategy, the model generates errors that significantly different from those expected for realistic output, resulting in suboptimal performance due to the associated loss. We hypothesize that the full strategy's performance of SRL is even worse than that observed in NLI, due to the significantly larger output space.

To summarize, sampling strategy and projection-con, projection-both mechanisms consistently demonstrate superior performance across all tasks. However, in relation to the type of constraint loss, there is no type of loss that consistently shows superior performance across all tasks; it varies depending on the task. As shown in Table 3, the number of combinations of learning algorithms with output constraints based on our analysis criteria is quite large (65 combinations for the NLI and SRL tasks,

[1]For example, $\underset{\pm 00.77}{84.72}$ means that the average is 84.72, and the standard deviation is 00.77 from 10 experiments.

[2]Constraint violation rate is used for table 3. However, when we consider $H\beta$-score, we convert it to the constraint satisfaction rate, which is $1-$(constraint violation rate).

and 40 combinations for the STE task). Therefore, it is practically impossible to experiment with all learning algorithms. We examined the performance trends of the algorithms through figures 1, 2, and 3, which provide useful insights for selecting learning algorithms.

**Specific combinations of axes outperforming others** In addition to observing overall trends, we dive into a more detailed analysis of specific algorithm combinations and their performance. We observed in the previous experimental results (figures 4, 5, 6) that the sampling strategy and projection-con, projection-both mechanisms generally perform well, with performance improving as sample size increases. However, the results in figures 4, 5, and 6 represent averages across multiple algorithm outcomes and do not depict individual algorithms. In this section, we narrow our focus and present an analysis for individual algorithms assuming a fixed sampling strategy with a sample size of 10 (referred to as samp-10 from now on) which consistently performed the best across different tasks, across different conditions. Figures 1, 2, and 3 depict the $H\beta$-scores for different combinations of loss types and integration mechanisms when the sampling strategy is fixed as samp-10. For visibility, we consider values of 0.3, 1, and 3.

Notably, our newly proposed projection-based algorithms, projection-con and projection-both, exhibit the highest-level performance across most situations. One interesting point is the performance difference between projection-con and projection-both mechanisms. By examining the average of the top 5 number of $H\beta$-scores (as previously shown in figure 4), we find that projection-con outperforms other mechanisms. However, upon observing individual algorithms per task, we found that for the soft or real types of loss, the projection-both mechanism shows the best-level performance than other mechanisms for most combinations. In the case of the SRL task, there are instances where the monotone mechanism performs well. Particularly, when used in conjunction with a soft type of loss, the monotone mechanism exhibits higher performance, which is inconsistent with other experimental results. The reason for this discrepancy has not been clearly identified yet, but the specific characteristics of weight updates in constraint loss combined with a soft type of loss for achieving higher performance remain a subject for future work.

Another noteworthy observation is that under samp-10, the soft type of loss exhibits the highest performance in most cases. Results from figures 1 and 3 show that, except for the projection-sup instance in SRL, soft type of loss generally outperforms real type of loss. We previously observed from figure 5 that real type of loss tends to perform best in the SRL task among the three types of losses. The results in 3, however, demonstrate the opposite, indicating that soft type of loss performs exceptionally well when combined with the sampling strategy.

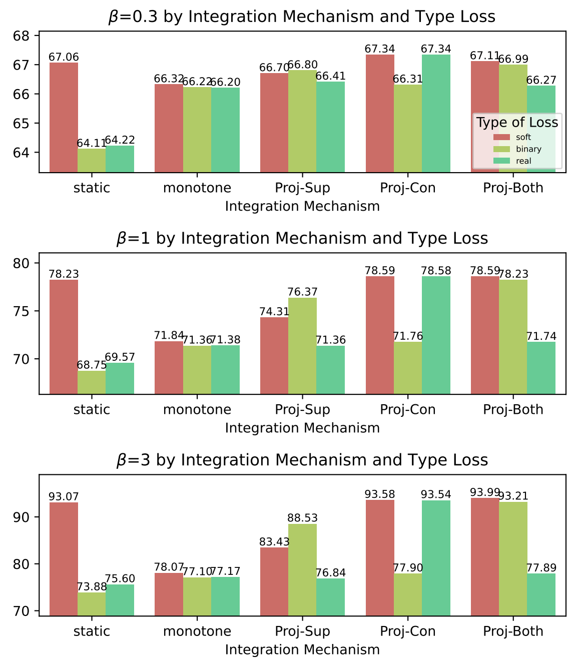

Figure 1: Experiment result from NLI task with samp-10. Three bar plots represents the $H\beta$-scores with respect to the integration mechanism (separated by the $x$-axis) and type of constraint losses (separated by the color). From top to bottom, the corresponding values of $\beta$'s are 0.3, 1, 3, respectively.

## 6 Additional related work

There are two stages where constraints can be injected: at inference time and learning time. At inference time, the goal is to remedy nonsensical outputs that violate human constraints at test time regardless of the training procedure (Lee et al., 2019; Roth and Yih, 2005). For example, (Lee et al., 2019) updates the model parameters at test time for each test instance to satisfy constraints, while (Roth and Yih, 2005) transforms the constrained problem into the form of integer linear programming for the inference process to maximize the log probability score with constraint satisfaction. Both

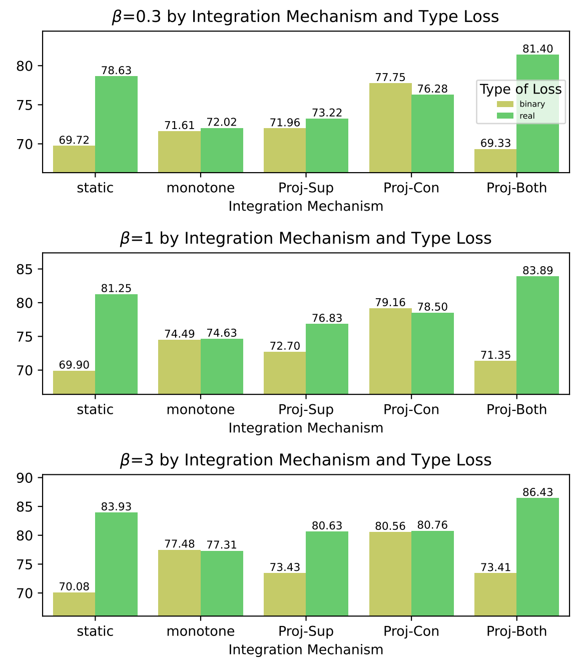

Figure 2: Experiment result from STE task with samp-10. Three bar plots represents the $H\beta$-scores with respect to the integration mechanism (separated by the $x$-axis) and type of constraint losses (separated by the color). From top to bottom, the corresponding values of $\beta$'s are 0.3, 1, 3, respectively.

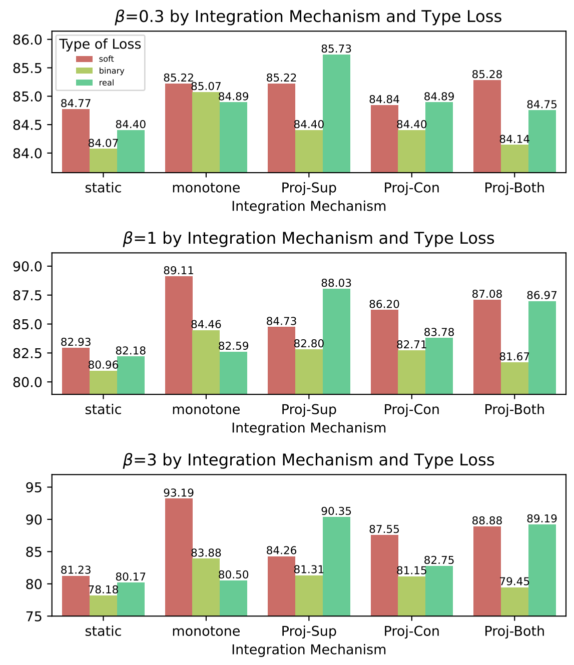

Figure 3: Experiment result from SRL task with samp-10. Three bar plots represents the $H\beta$-scores with respect to the integration mechanism (separated by the $x$-axis) and type of constraint losses (separated by the color). From top to bottom, the corresponding values of $\beta$'s are 0.3, 1, 3, respectively.

methods have shown to satisfy constraints well and also improve performances, however, as these two methods utilize vastly different philosophies, their formulations are not directly comparable.

On the other hand, another line of research have explored how to practically use constraint injection in software development (Ahmed et al., 2022; Rajaby Faghihi et al., 2021), which demonstrate the application of constraint injection techniques in software. These tools can apply constraints across different domains, highlighting the versatility of constraint learning methods. Importantly, they are effective not only during the training phase but also during inference. This research and the platforms mentioned offer valuable insights into the practical application of constraints.

## 7 Conclusions

We have proposed three axes for classifying and categorizing learning algorithms related to injecting constraints: type of constraint loss, exploring strategy of constraint-violating examples, and integration of main task and constraint information. To the best of our knowledge, this study is the first to systematically classify existing learning algorithms with constraints under a unified formulation. We have analyzed the key factors that affect performance based on our analysis criteria, which helps in understanding learning algorithms with constraints.

Additionally, we have introduced three projection-based mechanisms as a novel approach for the integration mechanism of main task and constraint information. Viewing the main task and constraint injection as two separate tasks, we started with the motivation to prevent negative effects on each other during the gradient update process. This introduces a new perspective on integrating learning signals from main task and constraint, which shows superior performance compared to existing integration mechanisms.

## 8 Limitations and future work

Our experiments were exclusively conducted on NLP tasks (NLI, STE, SRL) and did not include cutting-edge large language models. Therefore, it would be worthwhile to extend our experiments to a broader range of tasks and larger models. This would not only validate the generalizability of our methods but also potentially uncover new insights and improvements for various applications.

## References

Kareem Ahmed, Tao Li, Thy Ton, Quan Guo, Kai-Wei Chang, Parisa Kordjamshidi, Vivek Srikumar, Guy Van den Broeck, and Sameer Singh. 2022. Pylon: A pytorch framework for learning with constraints. In *NeurIPS 2021 Competitions and Demonstrations Track*, pages 319–324. PMLR.

Michał Baczyński and Balasubramaniam Jayaram. 2007. On the characterizations of (s, n)-implications. *Fuzzy sets and systems*, 158(15):1713–1727.

Benjamín Callejas Bedregal, Graçaliz Pereira Dimuro, Regivan Hugo Nunes Santiago, and Renata Hax Sander Reiser. 2010. On interval fuzzy s-implications. *Information Sciences*, 180(8):1373–1389.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

Matthias Bröcheler, Lilyana Mihalkova, and Lise Getoor. 2010. Probabilistic similarity logic. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, UAI'10, page 73–82, Arlington, Virginia, USA. AUAI Press.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. 2018. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*.

Jay Yoon Lee, Sanket Vaibhav Mehta, Michael Wick, Jean-Baptiste Tristan, and Jaime Carbonell. 2019. Gradient-based inference for networks with output constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4147–4154.

Tao Li, Parth Anand Jawale, Martha Palmer, and Vivek Srikumar. 2020. Structured tuning for semantic role labeling. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8402–8412, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

David Lopez-Paz and Marc'Aurelio Ranzato. 2017. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30.

Sanket Vaibhav Mehta, Jay Yoon Lee, and Jaime Carbonell. 2018. Towards semi-supervised learning for deep semantic role labeling. *arXiv preprint arXiv:1808.09543*.

Pasquale Minervini and Sebastian Riedel. 2018. Adversarially regularising neural NLI models to integrate logical background knowledge. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 65–74, Brussels, Belgium. Association for Computational Linguistics.

Yatin Nandwani, Abhishek Pathak, and Parag Singla. 2019. A primal dual formulation for deep learning with constraints. *Advances in Neural Information Processing Systems*, 32.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. 2012. Conll-2012 shared task: Modeling multilingual unrestricted coreference in ontonotes. In *Joint conference on EMNLP and CoNLL-shared task*, pages 1–40.

Hossein Rajaby Faghihi, Quan Guo, Andrzej Uszok, Aliakbar Nafar, and Parisa Kordjamshidi. 2021. DomiKnowS: A library for integration of symbolic domain knowledge in deep learning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 231–241, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Hossein Rajaby Faghihi, Aliakbar Nafar, Chen Zheng, Roshanak Mirzaee, Yue Zhang, Andrzej Uszok, Alexander Wan, Tanawan Premsri, Dan Roth, and Parisa Kordjamshidi. 2023. Gluecons: A generic benchmark for learning under constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9552–9561.

Dan Roth and Wen-tau Yih. 2005. Integer linear programming inference for conditional random fields. In *Proceedings of the 22nd international conference on Machine learning*, pages 736–743.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Broeck. 2018. A semantic loss function for deep learning with symbolic knowledge. In *International conference on machine learning*, pages 5502–5511. PMLR.

## A More specific comparison between types of constraint losses: PSL and REINFORCE

In this section, we dive deeper into the characteristics and applicability of the two types of constraint loss mentioned in Section 2.1: Probabilistic Soft Logic (PSL) and REINFORCE. These types of losses have unique strengths and weaknesses depending on the different circumstances.

## A.1 Fine-grained expressiveness of constraints

PSL stands superior in capturing more fine-grained information compared to REINFORCE. The PSL type of loss function evaluates not only the overall outcome but also performance of individual components to encourage more detailed feedback. For instance, consider a multi-label classification setting, where a particular book's every possible category needs to be predicted. An easily understandable example of constraint is associated with the hierarchical structure between labels: If a model predicts 'science fiction', it must necessarily also make a prediction that includes a hierarchy higher than that, which is 'fiction'. Note that, the above hierarchical constraint can be considered in the form of conditional statement for propositions, as follows:

$$\text{Pred(science fiction)} \implies \text{Pred(fiction)}$$

For the sake of simplicity in explanation, we employ the Łukasiewicz logic for this example. The soft value corresponding to the above logical expression is:

$$\min(1, 1 - P_s + P_f) \tag{7}$$

, where $P_s$ and $P_f$ represent the probability of being predicted for the science fiction class, and fiction class, respectively. In constraint learning using PSL the learning process aims to increase the soft value (7). In this example, the learning is conducted to increase $P_f - P_s$. Since 'fiction' is a class with higher hierarchy, and more inclusive class than 'science fiction', learning to increase $P_f - P_s$ is highly reasonable. Likewise, the PSL type of constraint loss can enrich the model's understanding and providing more detailed feedback. In REINFORCE, however, if the model's prediction violates the constraint, the probability for the prediction is directly reflected in the loss, regardless of the constraint imposed. This makes it challenging to provide detailed information about specifically which part should we penalize in the model's prediction. To incorporate more fine-grained information in REINFORCE, there is research that utilizes real rewards. For example, Mehta et al. defines reward score as $s = 1 - 2g \in [-1, 1]$, where $g \in [0, 1]$ stands for normalized error count, so that larger constraint violations lead to greater constraint loss. Although the loss function cannot reflect the soft value of logical expression, by assigning rewards differently based on the degree of constraint violation, it is possible to incorporate more fine-grained information into the loss function than just assigning binary rewards.

| Logic | Product | Gödel | Łukasiewicz |
|---|---|---|---|
| Negation | $1-a$ | $1-a$ | $1-a$ |
| T-conorm | $a+b-ab$ | $\max(a,b)$ | $\min(1,a+b)$ |
| T-norm | $ab$ | $\min(a,b)$ | $\max(0,a+b-1)$ |
| Implication | $\begin{cases}1 & \text{if } a \le b \\ b/a & \text{otherwise}\end{cases}$ | $\begin{cases}1 & \text{if } a \le b \\ b & \text{otherwise}\end{cases}$ | $\min(1,1-a+b)$ |

Table 1: Examples of logics. Our experiment used Gödel logic except for the implication (⇒). For ⇒, we used S-implication (Baczyński and Jayaram, 2007; Bedregal et al., 2010) form, $\max(1-a, b)$.

## A.2 Type of constraints that constraint loss can represent

Though PSL stands superior in capturing more fine-grained information compared to REINFORCE, PSL encounters difficulties when representing a variety of constraints, while REINFORCE can express arbitrary types of constraints. Rajaby Faghihi et al. introduced limitations in encoding a specific type of knowledge in research related to constraint injection during training time (Nandwani et al., 2019; Ahmed et al., 2022). Instead of focusing on individual characteristics of these studies, we can generalize this limitation using our view of analysis axes. As in table 2, we can rewrite the constraint types that each constraint loss type can handle, according to the two types of constraint loss: PSL and REINFORCE.

NC (Needs Conversion) in table 2 can sometimes be practically challenging due to significant overhead, making it difficult to leverage effectively. An example is seen in the STE task discussed in §4, where the constraint is defined as follows: the count of 'b' in the target should be exactly three times that in the source. Let $s \in (az|bz)^*$ be an input sequence data, and $t$ be a predicted output sequence of model. Also, for a finite set $X = \{x_1, x_2, \cdots, x_n\} \subseteq \mathbb{N}$, let $s(X)$, $t(X)$ represent the presence of 'b' at positions $x_1, ..., x_n$ in the sequences $s$ and $t$, respectively. Note that if the count of 'b' in $s$ is 1, then the count of 'b' in $t$ should be 3. While this represents a small portion of the original constraint, when expressed in the linearized logical expression for PSL application, it can be represented as follows:

$$\bigwedge_{i=1}^{|s|} \left[ s(\{i\}) \implies \bigvee_{1 \le j_1 < j_2 < j_3 \le |t|} t(\{j_1, j_2, j_3\}) \right]$$

|  | Seq | Lin | Log | Log+Quan | Prog |
|---|---|---|---|---|---|
| PSL | ✓ | ✓ | NC | NC | X |
| REINFORCE | ✓ | ✓ | ✓ | ✓ | ✓ |

Table 2: This table classifies constraints that constraint-injection methods can handle during training time. We reinterpret table 2 of (Rajaby Faghihi et al., 2023) with our axes of analysis: type of constraint loss. The specific meaning of abbreviations are as follows: Seq=sequential structure, Lin=linear constraint, Log=logical constraint, Log+Quan=logical constraint with quantifier, Prog=any constraints encoded as a program, NC=needs conversion.

However, even this partial inclusion of the overall constraint requires an excessively high computational cost.

The comparison between two types of constriant losses illustrates that the appropriate type of loss may vary depending on task requirements and problem details, reflecting the inevitable trade-off between the level of detailed information about constraints and the scope of constraint representation.

## B Experiment result

Table 3 represents the experiment results for all combinations, containing main task metrics(%, denoted as "Perf") and constraint violation rates(%, denoted as "Const.vio"). SRL, NLI, and STE tasks used F1 score, accuracy, and token accuracy for main task metric, respectively. For the types of constraint losses, soft, binary, and real respectively represents PSL, REINFORCE method with binary reward, and REINFORCE method with real reward. The term 'Baseline' refers to the experiment results without any constraint injection. Ahmed et al., using the REINFORCE - binary reward method, separates the generation of constraint loss into two approaches in their implementation[3]: one for decoded samples that satisfy the constraints and another for those that do not. In Mehta et al., 2018, they generates constraint loss for decoded samples only when the constraints are violated. To compare various algorithms under a unified formulation, experiments involving constraint loss related to REINFORCE were conducted by generating constraint loss for examples that violated the constraints.

Note that we can easily extend the learning algorithms with constraints to semi-supervised learning. For SRL and NLI tasks, we also utilized unlabeled data during the training process. For SRL, we randomly selected 3% of training data for unlabeled data. For NLI, we utilized the unlabeled data used in (Ahmed et al., 2022)[4].

[3]https://github.com/pylon-lib/pylon

**Algorithm 1** Pseudo code for projection-both mechanism

**Input**: labeled data $\mathcal{D}_L = \langle x_i, y_i \rangle_{i=1}^T$, unlabeled data $\mathcal{D}_U = \langle x_i^u \rangle_{i=1}^T$ (if available), model parameter $\theta$.

```
1: Initialize: $g_{sup}^{ref} \leftarrow 0$, $g_{con}^{ref} \leftarrow 0$.
2: while not converge do
3:     $\langle x_L, y_L \rangle \leftarrow$ sample from $\mathcal{D}_L$
4:     $\langle x_U \rangle \leftarrow$ sample from $\mathcal{D}_U$
5:     $g_{sup} \leftarrow \nabla \mathcal{L}(x_L, y_L; \theta)$
6:     $g_{con} \leftarrow \nabla(\mathcal{C}(x_L; \theta) + \mathcal{C}(x_U; \theta))$
7:     if $g_{sup} \cdot g_{con}^{ref} < 0$ then
8:         $g_{sup} \leftarrow$ project $g_{sup}$ via $g_{con}^{ref}$
9:     end if
10:    if $g_{con} \cdot g_{sup}^{ref} < 0$ then
11:        $g_{con} \leftarrow$ project $g_{con}$ via $g_{sup}^{ref}$
12:    end if
13:    $g_{sup}^{ref} \leftarrow$ store the averaged vector of $g_{sup}$ across gradient updates.
14:    $g_{con}^{ref} \leftarrow$ store the averaged vector of $g_{con}$ across gradient updates.
15:    Gradient update of $\theta$ for the cumulative gradients: $g_{sup}$ and $g_{con}$.
16: end while
```

## C Pseudo-code for projection based integration mechanism

Algorithm 1 shows the detail pseudo-code for *projection-both* mechanism. Pseudo-codes for projection-sup and projection-con mechanisms are variant of algorithm 1. For projection-sup, there is no need to store $g_{con}^{ref}$, nor to calculate the dot product between $g_{sup}$ and $g_{con}^{ref}$. Likewise, for projection-con, there is no need to store $g_{sup}^{ref}$, nor to calculate the dot product between $g_{con}$ and $g_{sup}^{ref}$.

## D Additional details about selected tasks and implementations

### D.1 SRL

The baseline employs the RoBERTa baseline model (Liu et al., 2019), and two linear layers are added

[4]https://github.com/pylon-lib/pylon/tree/master/examples/nli

after the last layer of RoBERTa. While the parameters of the RoBERTa model are fixed, only the parameters of the last two linear layers are trained.

The model predicts one of 9 tags - {`O`, `BO`, `IO`, ..., `B3`, `I3`} - and transforms all other tags into `O`. During the training process, 3% of the data is randomly sampled from the training data for use.

For the real type of constraint loss in REINFORCE algorithm, the method employed to assign rewards is based on the count of duplicates in `B`. For all types of `B-X` that appear more than once, we summed the occurrences of all number of constraint-violated `B-X` and divided by the total sequence length, and this is multiplied by the constraint loss.

### D.2 NLI

The baseline employs the RoBERTa baseline model (Liu et al., 2019), and two linear layers are added after the last layer of RoBERTa. While the parameters of the RoBERTa model are fixed, only the parameters of the last two linear layers are trained. During training, 20% of the training data is randomly sampled for use.

For the real type of constraint loss in REINFORCE algorithm, the method employed to assign rewards is based on the value in the PSL. In cases where it violates constraints, the corresponding PSL values are multiplied by the constraint loss.

### D.3 STE

The training data includes 3 to 6 instances of az' and bz' in the source language, generating a dataset of 6000 instances. The test data comprises 3 to 8 instances of 'az' and 'bz' in the source language, transformed into the target language. We utilize seq2seq (Sutskever et al., 2014) LSTM for prediction.

For the real type of constraint loss in REINFORCE algorithm, the method employed to assign rewards is a length-normalized quadratic: $(3x_b - y_b)^2/(\text{len}(x) + \text{len}(y))$, where $x$ and $y$ respectively represents the input and output, while $x_b$ and $y_b$ respectively represents the number of occurrences of 'b' in the input and output.

We don't utilize a PSL type of constraint loss for STE task. This is because expressing constraints about the number of 'b' occurrences in input and output is highly intricate for PSL. This constraint serves as an example demonstrating the difficulty of applying PSL to all types of constraints.

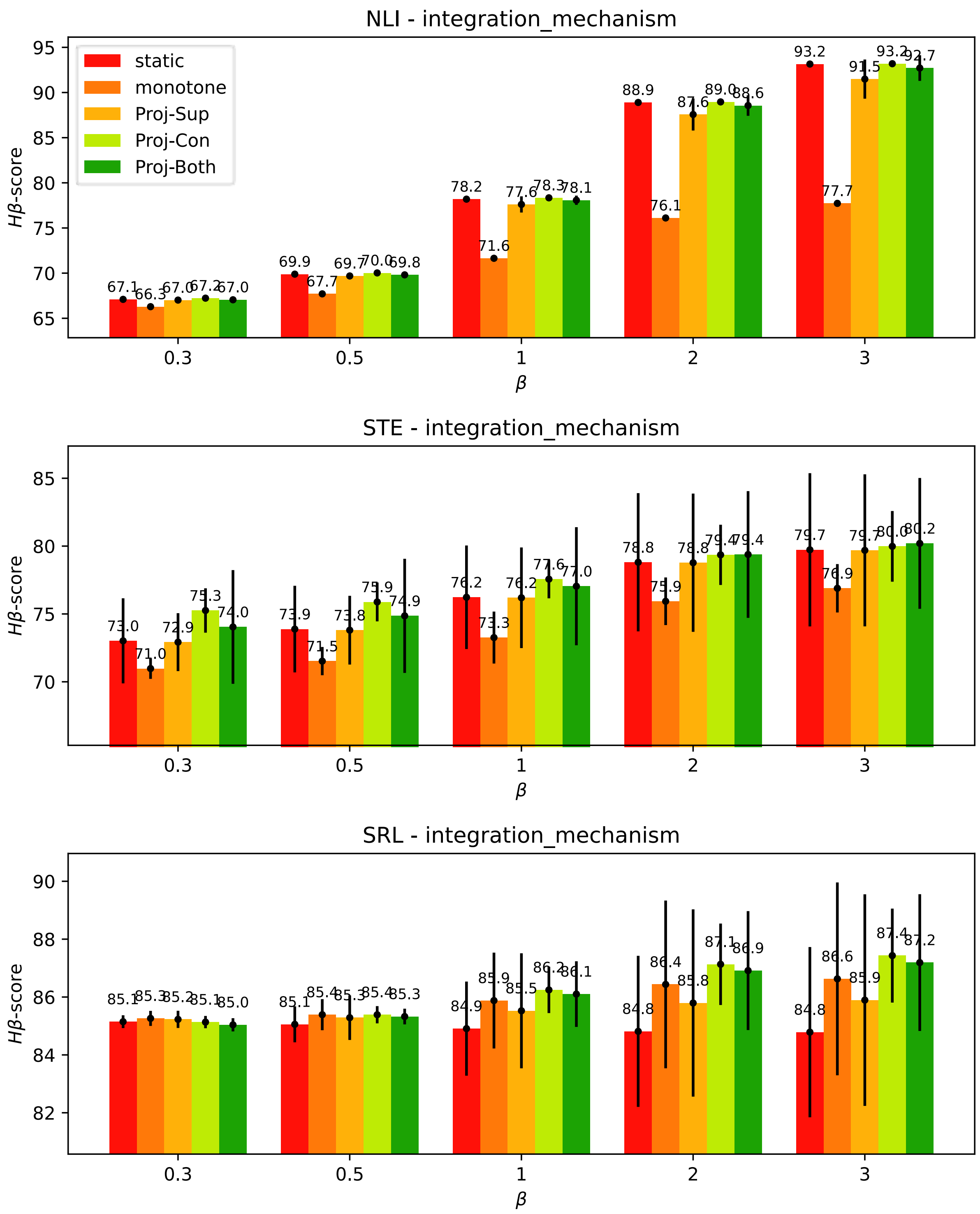

Figure 4: The $H\beta$-score values for different values of $\beta$ for three tasks: NLI, STE and SRL. For each $\beta$, the top 5 experimental results with the highest $H\beta$-scores are presented for each of the five integration mechanisms described in section §2.3.

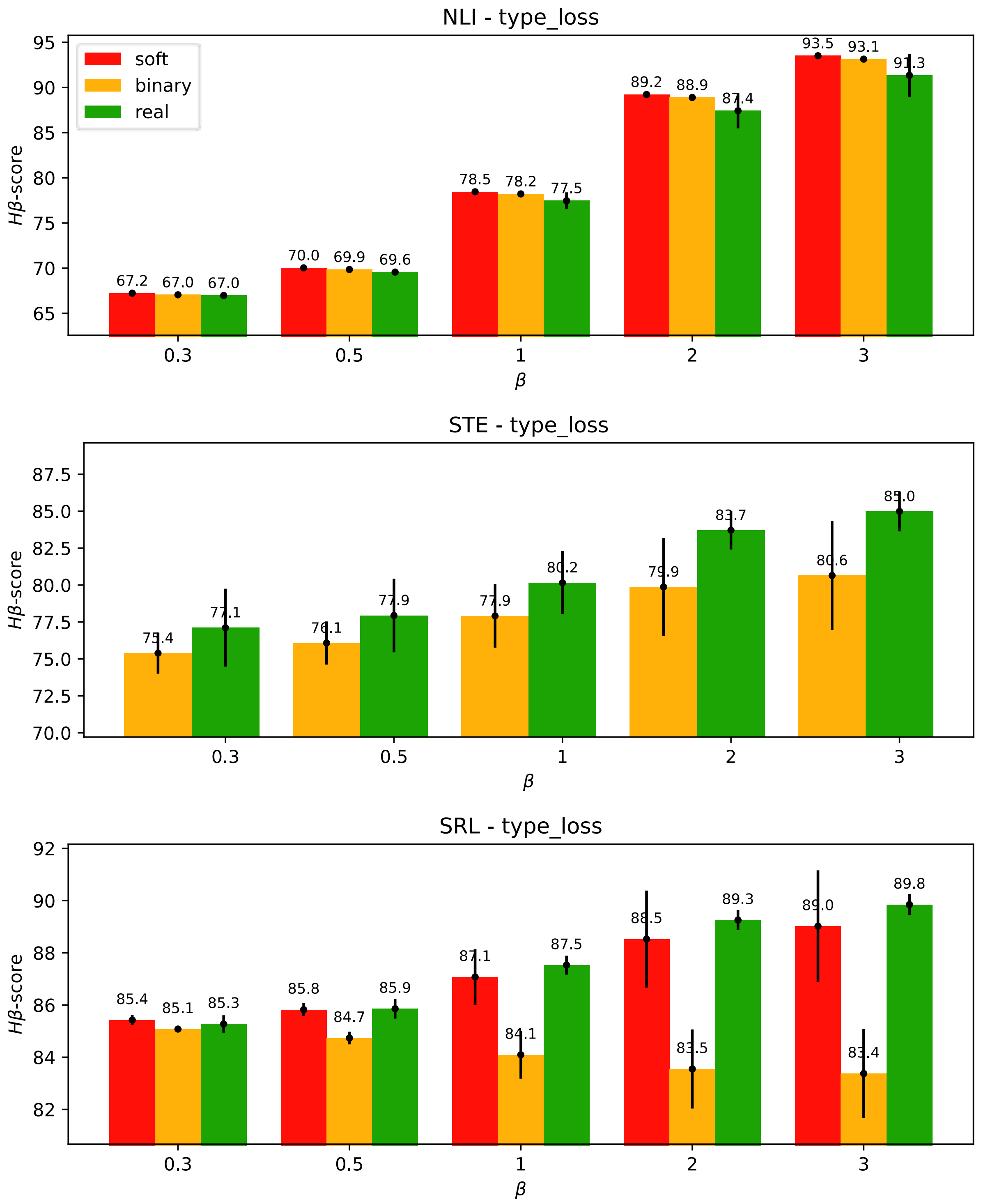

Figure 5: The $H\beta$-score values for different values of $\beta$ for three tasks: NLI, STE and SRL. For each $\beta$, the top 5 experimental results with the highest $H\beta$-scores are presented for each of the three types of constraint losses described in section §2.1.

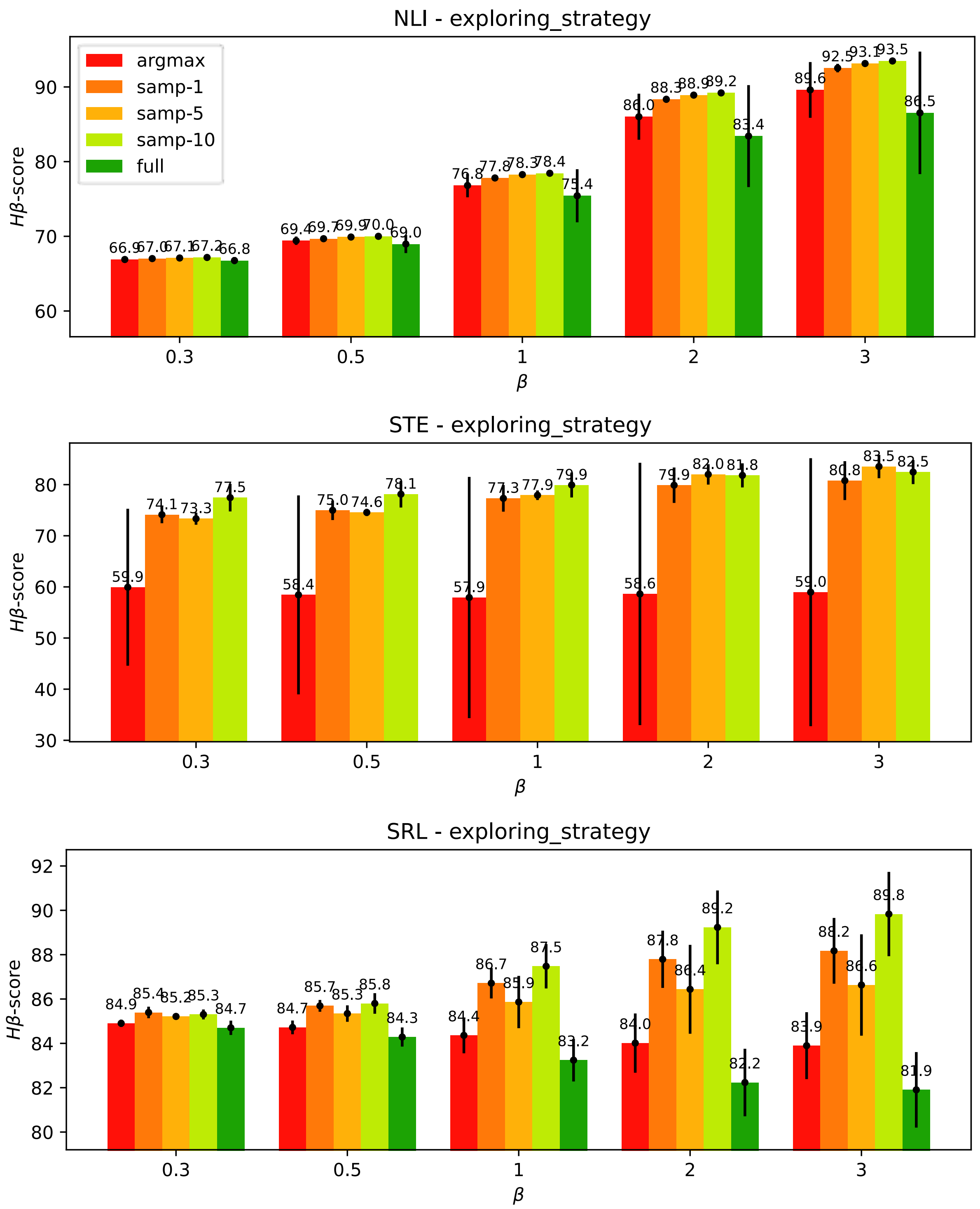

Figure 6: The $H\beta$-score values for different values of $\beta$ for three tasks: NLI, STE and SRL. For each $\beta$, the top 5 experimental results with the highest $H\beta$-scores are presented for each of the five exploring strategies described in section §2.2.

| Task | | | Top-1 | | Sampling-1 | | Sampling-5 | | Sampling-10 | | Exhaustive | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Perf | Const.Vio | Perf | Const.Vio | Perf | Const.Vio | Perf | Const.Vio | Perf | Const.Vio |
| NLI | Baseline | | Acc: 65.14 ±00.30, Const.Vio: 20.81 ±02.57 | | | | | | | | | |
| | soft | static | 65.18 ±00.35 | 04.72 ±00.88 | 65.40 ±00.40 | 03.60 ±00.60 | 65.31 ±00.26 | 02.23 ±00.36 | 65.22 ±00.40 | 02.29 ±01.88 | 65.20 ±00.34 | 01.95 ±00.22 |
| | | monotone (λ ↑) | 65.21 ±00.26 | 21.51 ±03.03 | 65.20 ±00.27 | 20.24 ±02.09 | 65.30 ±00.38 | 21.83 ±01.74 | 65.33 ±00.54 | 20.20 ±01.67 | 65.20 ±00.55 | 22.72 ±02.05 |
| | | Proj-Sup | 65.28 ±00.43 | 20.14 ±02.99 | 65.05 ±00.48 | 20.73 ±01.61 | 65.26 ±00.40 | 02.08 ±00.38 | 65.38 ±00.20 | 13.93 ±02.55 | 65.36 ±00.49 | 01.95 ±00.41 |
| | | Proj-Con | 65.46 ±00.27 | 03.05 ±00.42 | 65.23 ±00.40 | 03.54 ±00.74 | 65.05 ±00.26 | 02.40 ±00.33 | **65.48** * **±00.28** | 01.73 ±00.17 | 65.30 ±00.29 | 02.50 ±00.36 |
| | | Proj-Both | 65.20 ±00.40 | 17.45 ±03.13 | 65.41 ±00.29 | 06.16 ±01.75 | 65.39 ±00.21 | 08.19 ±05.04 | 65.23 ±00.32 | **01.17** * **±00.84** | 65.40 ±00.43 | 21.30 ±03.29 |
| | binary | static | 65.20 ±00.38 | 02.45 ±02.65 | 64.23 ±00.44 | 03.24 ±05.29 | 63.26 ±00.38 | 17.02 ±11.67 | 63.26 ±00.49 | 24.72 ±02.40 | - | - |
| | | monotone (λ ↑) | 65.36 ±00.31 | 20.30 ±03.38 | 65.10 ±00.29 | 22.66 ±02.66 | 65.16 ±00.37 | 22.00 ±01.78 | 65.29 ±00.36 | 21.32 ±02.74 | - | - |
| | | Proj-Sup | 65.21 ±00.35 | 14.73 ±03.30 | 65.25 ±00.47 | 07.06 ±01.66 | 65.22 ±00.23 | 02.25 ±00.20 | 65.18 ±00.25 | 07.80 ±04.97 | - | - |
| | | Proj-Con | 65.11 ±00.49 | 07.49 ±06.08 | **65.42 ±00.19** | 11.80 ±02.43 | 65.25 ±00.35 | 02.57 ±00.16 | 65.33 ±00.24 | 20.40 ±03.01 | - | - |
| | | Proj-Both | 65.16 ±00.28 | 19.76 ±02.85 | 65.05 ±00.34 | 02.11 ±00.28 | 65.23 ±00.41 | **02.10 ±00.37** | 65.14 ±00.49 | 02.18 ±00.33 | - | - |
| | real | static | 65.26 ±00.24 | 20.60 ±01.29 | 64.66 ±00.34 | 01.92 ±00.90 | 63.28 ±00.44 | 23.82 ±01.89 | 63.26 ±00.30 | 22.72 ±07.16 | - | - |
| | | monotone (λ ↑) | 65.42 ±00.21 | 21.45 ±01.88 | 65.14 ±00.49 | 20.97 ±02.25 | 65.26 ±00.38 | 21.36 ±02.26 | 65.26 ±00.38 | 21.23 ±03.68 | - | - |
| | | Proj-Sup | 65.26 ±00.26 | 22.93 ±01.96 | 65.21 ±00.25 | 22.70 ±02.23 | 65.11 ±00.39 | 20.68 ±02.76 | **65.51 ±00.48** | 21.65 ±02.03 | - | - |
| | | Proj-Con | 65.27 ±00.32 | 20.98 ±01.67 | 65.33 ±00.36 | 08.75 ±01.86 | 65.04 ±00.24 | 02.39 ±00.36 | 65.49 ±00.38 | **01.78 ±00.20** | - | - |
| | | Proj-Both | 65.09 ±00.34 | 23.21 ±01.27 | 65.15 ±00.31 | 07.04 ±02.77 | 65.40 ±00.39 | 09.33 ±03.42 | 65.29 ±00.26 | 20.40 ±03.62 | - | - |
| STE | Baseline | | Tok-Acc: 67.26 ±02.26, Const.Vio: 28.89 ±10.07 | | | | | | | | | |
| | binary | static | 69.98 ±00.43 | 29.35 ±12.11 | 73.59 ±02.37 | 18.83 ±12.72 | 69.35 ±02.20 | 33.32 ±06.73 | 69.68 ±03.22 | 29.87 ±17.58 | - | - |
| | | monotone (λ ↑) | 67.17 ±06.42 | 26.93 ±13.09 | 71.03 ±05.72 | 33.55 ±11.46 | 69.43 ±03.79 | 24.89 ±11.15 | 71.07 ±04.63 | 21.74 ±08.34 | - | - |
| | | Proj-Sup | 43.55 ±04.03 | 93.87 ±03.65 | 75.01 ±05.90 | **10.86 ±07.28** | 69.19 ±03.67 | 26.21 ±16.22 | 71.81 ±02.96 | 26.38 ±17.69 | - | - |
| | | Proj-Con | 49.64 ±03.27 | 98.40 ±02.10 | 73.58 ±03.97 | 22.18 ±16.16 | 74.96 ±06.76 | 22.71 ±16.42 | **77.48 ±08.92** | 19.08 ±14.79 | - | - |
| | | Proj-Both | 51.19 ±01.70 | 98.77 ±02.08 | 70.71 ±03.92 | 23.43 ±12.72 | 68.51 ±03.34 | 26.62 ±10.77 | 68.94 ±02.20 | 26.06 ±15.75 | - | - |
| | real | static | 67.77 ±04.25 | 30.18 ±12.86 | 70.16 ±03.03 | 22.18 ±12.04 | 70.30 ±04.02 | **10.86** * **±07.53** | 78.13 ±05.00 | 15.37 ±11.15 | - | - |
| | | monotone (λ ↑) | 63.73 ±04.43 | 22.92 ±13.45 | 60.74 ±02.93 | 51.79 ±34.51 | 69.91 ±03.29 | 19.07 ±13.41 | 71.53 ±03.74 | 21.99 ±13.05 | - | - |
| | | Proj-Sup | 46.30 ±04.91 | 94.17 ±04.92 | 54.32 ±02.47 | 98.86 ±01.72 | 73.02 ±04.14 | 15.04 ±08.75 | 72.55 ±02.49 | 18.36 ±10.61 | - | - |
| | | Proj-Con | 48.20 ±03.88 | 95.29 ±04.58 | 52.98 ±01.16 | 99.74 ±00.73 | 72.24 ±03.04 | 14.26 ±07.57 | 75.86 ±03.16 | 18.66 ±08.36 | - | - |
| | | Proj-Both | 50.60 ±02.60 | 98.68 ±03.16 | 74.81 ±05.59 | 17.40 ±16.24 | 72.00 ±04.31 | 15.03 ±14.56 | **80.92** * **±04.24** | 12.91 ±06.07 | - | - |
| SRL | Baseline | | F1: 84.72 ±00.77, Const.Vio: 20.43 ±04.09 | | | | | | | | | |
| | soft | static | 85.24 ±01.49 | 15.17 ±02.04 | 85.02 ±01.55 | 14.07 ±03.02 | 85.21 ±01.13 | 19.53 ±02.80 | 85.15 ±00.74 | 19.18 ±36.95 | 85.31 ±00.98 | 21.72 ±04.61 |
| | | monotone (λ ↑) | 84.42 ±01.07 | 18.53 ±04.44 | **85.78** * **±01.46** | 14.40 ±03.66 | 85.12 ±01.23 | 16.38 ±03.12 | 84.49 ±01.16 | **05.73** * **±01.42** | 85.18 ±00.81 | 15.97 ±02.97 |
| | | Proj-Sup | 85.02 ±00.98 | 20.79 ±04.63 | 85.19 ±01.24 | 20.23 ±04.85 | 85.09 ±01.03 | 19.59 ±03.93 | 85.32 ±01.26 | 15.86 ±04.08 | 85.18 ±00.97 | 18.73 ±04.03 |
| | | Proj-Con | 84.96 ±00.60 | 15.72 ±01.23 | 85.24 ±01.53 | 11.76 ±02.44 | 85.07 ±00.90 | 12.80 ±02.85 | 84.58 ±01.17 | 12.11 ±03.61 | 84.31 ±00.57 | 18.04 ±04.01 |
| | | Proj-Both | 85.21 ±01.21 | 21.86 ±03.13 | 84.71 ±01.06 | 12.11 ±00.37 | 85.62 ±01.68 | 17.68 ±03.57 | 84.93 ±02.06 | 10.66 ±01.83 | 85.03 ±01.11 | 17.64 ±04.26 |
| | binary | static | 85.20 ±01.51 | 19.84 ±00.66 | 85.52 ±01.02 | 19.53 ±02.34 | 85.19 ±00.91 | 18.90 ±02.91 | 84.71 ±01.28 | 22.48 ±04.34 | - | - |
| | | monotone (λ ↑) | 84.51 ±00.94 | **14.25 ±02.93** | 84.36 ±00.75 | 20.98 ±05.88 | 85.23 ±01.44 | 15.50 ±03.37 | 85.19 ±00.68 | 16.26 ±04.40 | - | - |
| | | Proj-Sup | 84.14 ±01.05 | 21.20 ±02.12 | 84.57 ±01.20 | 19.99 ±02.20 | **85.70 ±01.01** | 22.44 ±03.92 | 84.73 ±00.57 | 19.05 ±03.03 | - | - |
| | | Proj-Con | 84.54 ±00.71 | 21.28 ±03.91 | 85.56 ±01.23 | 19.62 ±04.38 | 85.06 ±01.03 | 19.79 ±03.58 | 84.74 ±00.86 | 19.23 ±06.10 | - | - |
| | | Proj-Both | 85.23 ±00.77 | 20.36 ±04.22 | 84.89 ±01.36 | 19.14 ±03.33 | 84.85 ±01.014 | 19.46 ±03.91 | 84.61 ±00.63 | 21.09 ±03.39 | - | - |
| | real | static | 85.05 ±00.68 | 18.26 ±03.21 | 85.12 ±00.96 | 09.79 ±03.15 | 85.33 ±01.09 | 22.37 ±04.43 | 84.85 ±01.23 | 20.32 ±04.10 | | |
| | | monotone (λ ↑) | 84.73 ±00.78 | 21.99 ±04.21 | 85.09 ±00.96 | 23.78 ±03.95 | 84.94 ±01.29 | 19.61 ±04.50 | **85.36 ±01.14** | 20.00 ±02.65 | - | - |
| | | Proj-Sup | 85.19 ±01.46 | 19.19 ±04.13 | 85.09 ±00.90 | 17.16 ±04.40 | 84.87 ±00.94 | 09.21 ±02.60 | 85.29 ±01.25 | **09.05 ±02.20** | - | - |
| | | Proj-Con | 85.06 ±00.93 | 18.22 ±04.32 | 84.31 ±01.67 | 09.47 ±04.19 | 85.19 ±01.32 | 22.55 ±05.09 | 85.11 ±01.08 | 17.50 ±03.98 | - | - |
| | | Proj-Both | 85.05 ±00.94 | 18.54 ±04.55 | 85.25 ±00.74 | 20.36 ±04.28 | 84.54 ±00.41 | 11.96 ±02.87 | 84.33 ±07.19 | 10.23 ±03.35 | - | - |

Table 3: Experiment results for all combinations. The gray-colored numbers represent results with main task metrics and constraint violations worse than the baseline. For each type of constraint loss, results showing the highest main task metric and lowest constraint violation are highlighted in bold. For individual task, the highest main task metric and lowest constraint violation results are marked with an asterisk (*). In SRL and STE tasks, where the output takes the form of more than one token, the method of selecting the class with the highest probability for each token was employed for Top-1 strategy.

| | **NLI Rules** |
|---|---|
| R1 | T $\implies$ ent$(X_1, X_1)$ |
| R2 | con$(X_1, X_2)$ $\implies$ con$(X_2, X_1)$ |
| R3 | ent$(X_1, X_2)$ $\implies$ $\neg$con$(X_2, X_1)$ |
| R4 | neu$(X_1, X_2)$ $\implies$ $\neg$con$(X_2, X_1)$ |
| R5 | ent$(X_1, X_2) \wedge$ ent$(X_2, X_3)$ $\implies$ ent$(X_1, X_1)$ |

Table 4: NLI Rules in (Minervini and Riedel, 2018).