Revised ASBN Algorithm

Adversarial Sense Balance Network (training-time)

1 Notation (short)

j index for a token instance in the current batch.

w_j token type (word / subword) for token j.

$h_j \in \mathbb{R}^d$ contextual encoder vector for token j (output of DSCD encoder).

$p_j \in \Delta^{K_{w_j}-1}$ DSCD soft distribution over K_{w_j} prototypes for type w_j.

$\hat{y}_j = \arg\max_k p_{j,k}$ DSCD predicted sense id for token j.

$p_{\max}(j) = \max_k p_{j,k}$ model confidence (top probability).

$U_j \in [0, 1\ ] combineduncertaintyscorefortoken$j$(entropy, MC - var,$$\sigma$-Net, novelty normalized).

$g_j \in \{0, 1\}$ DSCD gate / flag (1 if token is flagged ambiguous and requires ASBN attention).

F_w empirical type-level sense frequency vector for token type w from training corpus.

$z^{\text{tgt}}_{a(j)}$ aligned target-side embedding for the source token j (when available from parallel data).

$D_{\mathbf{freq}}, D_{\mathbf{ctx}}, D_{\mathbf{xl}}$ discriminators (frequency, context-sparse, cross-lingual) with parameters $\phi_{\text{freq}}, \phi_{\text{ctx}}, \phi_{\text{xl}}$.

$\bar{\lambda}_{\mathbf{freq}}, \bar{\lambda}_{\mathbf{ctx}}, \bar{\lambda}_{\mathbf{xl}}$ base GRL strengths for each discriminator.

$w_{\mathbf{freq}}, w_{\mathbf{ctx}}, w_{\mathbf{xl}}$ weights for discriminator losses in ASBN aggregation.

$\lambda_{\mathbf{ASBN}}$ global multiplier of ASBN loss in the full training objective.

$\text{clip}(\cdot, a, b)$ clip a scalar to the interval $[a, b]$.

2 ASBN compact pseudocode

Algorithm 1 ASBN: Training-step (per batch)

Require: Batch of tokens with DSCD outputs $\{h_j, p_j, \hat{y}_j, p_{\max}(j), U_j, g_j, F_{w_j}, z^{\text{tgt}}_{a(j)}\}$
Hyperparameters: $\bar{\lambda}_k$ (base GRL for $k \in \{\text{freq,ctx,xl}\}$), $\lambda_{\max}$, w_k, λ_{ASBN}.

1. Compute per-token GRL strengths (confidence-weighted).
for each token j in the batch **do**
  $\lambda_{k,j} \leftarrow \text{clip}\big(\bar{\lambda}_k \cdot p_{\max}(j) \cdot (1 - U_j) \cdot g_j,\ 0,\ \lambda_{\max}\big)$ ▷ $k \in \{\text{freq,ctx,xl}\}$
end for
2. Build discriminator inputs (via GRL).
for each token j **do**
  $x_{\text{freq},j} \leftarrow \text{concat}\big(\text{GRL}(h_j, \lambda_{\text{freq},j}),\ F_{w_j}\big)$
  $x_{\text{ctx},j} \leftarrow \text{concat}\big(\text{GRL}(h_j, \lambda_{\text{ctx},j}),\ \text{ctx\_stats}_j,\ U_j\big)$
  $x_{\text{xl},j} \leftarrow \text{concat}\big(\text{GRL}(h_j, \lambda_{\text{xl},j}),\ \text{proj}(z^{\text{tgt}}_{a(j)})\big)$
end for
3. Discriminator forward: compute per-token losses.
$L_{\text{freq}} \leftarrow \frac{1}{N}\sum_j \text{CE}\big(D_{\text{freq}}(x_{\text{freq},j}),\ y_j^{\text{freq}}\big)$
$L_{\text{ctx}} \leftarrow \frac{1}{N}\sum_j \text{CE}\big(D_{\text{ctx}}(x_{\text{ctx},j}),\ y_j^{\text{ctx}}\big)$
$L_{\text{xl}} \leftarrow \frac{1}{N}\sum_j \text{ContrastiveOrCE}\big(D_{\text{xl}}(x_{\text{xl},j}),\ z^{\text{tgt}}_{a(j)},\ \text{neg}_j\big)$
4. Aggregate ASBN loss.
$L_{\text{ASBN}} \leftarrow w_{\text{freq}} L_{\text{freq}} + w_{\text{ctx}} L_{\text{ctx}} + w_{\text{xl}} L_{\text{xl}}$
5. Update discriminators (minimize L_{ASBN}).
$\phi \leftarrow \phi - \eta_\phi \nabla_\phi L_{\text{ASBN}}$
6. Compute primary model losses (DSCD + NMT).
$L_{\text{MT}} \leftarrow$ translation loss (decoder)
$L_{\text{span}} \leftarrow$ BCE(span head)
$L_{\text{sense}} \leftarrow$ CE or contrastive sense loss
7. Update encoder & decoder (GRL supplies reversed gradients).
$L_{\text{total}} \leftarrow L_{\text{MT}} + \lambda_{\text{span}} L_{\text{span}} + \lambda_{\text{sense}} L_{\text{sense}} + \lambda_{\text{ASBN}} L_{\text{ASBN}}$
$\theta \leftarrow \theta - \eta_\theta \nabla_\theta L_{\text{total}}$
(Note: GRL negates & scales discriminator gradients en route to θ.)

3 Step-by-step explanation

1. Counterfactual Pair Generator (CPG). Creates artificial counterfactual examples to expose model shortcuts. *Example:* For the sentence "The bank is full of people", generate: (a) freq-swap: replace "bank" with less frequent sense (river bank); (b) context-lift: add clarifying phrase ("financial institution"); (c) xling-swap: deliberately mistranslate "bank" in the target. *Why:* Helps discriminators learn to catch frequency, context, and alignment biases.

2. Bank of Orthogonal Auditors. Multiple specialized critics instead of one adversary: - A-FREQ (frequency bias), - A-CTX (context insensitivity), - A-XL (cross-lingual mismatch), - A-PS (prototype stability auditor), - A-SC (spurious cue auditor, e.g. punctuation/digits). *Example:* If the model always picks "financial bank" regardless of context, A-FREQ flags it. *Why:* Allows targeted correction of different shortcut types.

3. Optimal-Transport Sense Balancer (OT-SB). Smooths predicted sense probabilities to avoid majority-sense collapse. *Example:* For "bass", if context is unclear, OT-SB enforces near-uniform distribution between fish and instrument senses. *Why:* Encourages fair treatment of senses when evidence is weak, while respecting strong context when available.

4. Primal–Dual No-Regret Multiplier Allocation. Adapts weights for each auditor dynamically. *Example:* If 70% of errors come from frequency bias, the system increases weight for A-FREQ during training. *Why:* Automatically shifts adversarial pressure to the shortcut causing most harm.

5. Editability Guard (E-GUARD). Checks whether the model's fixes are genuine semantic corrections or superficial hacks. *Example:* If "bass" is disambiguated only by looking at suffix "-s", E-GUARD penalizes it; if it uses deeper context like "guitar", E-GUARD rewards it. *Why:* Promotes durable, meaning-based corrections instead of surface tricks.

6. Retrieval-in-the-Loop (ARIL). Retrieves external glosses or examples for weak contexts. Includes an auditor that ensures the model does not overfit to the retrieval source. *Example:* For "jaguar", ARIL retrieves examples for both the animal and the car. Auditor ensures predictions depend on context, not retrieval source idiosyncrasies. *Why:* Strengthens disambiguation with external context while preserving robustness.

Below each numbered step from the pseudocode is explained: **what** the step does, **why** it is necessary, and **how** to implement it (practical tips).

Step 1: Compute per-token GRL strengths (confidence-weighted)

What For each token j and for each discriminator $k \in \{\text{freq}, \text{ctx}, \text{xl}\}$ compute a scalar $\lambda_{k,j}$ that controls how strongly the discriminator's gradient will be reversed when it propagates back into the encoder.

$$\lambda_{k,j} = \text{clip}\big(\bar{\lambda}_k \cdot p_{\max}(j) \cdot (1 - U_j) \cdot g_j,\ 0,\ \lambda_{\max}\big).$$

Why - We want stronger adversarial pressure when the model is *confident* ($p_{\max}$ high) but *not genuinely uncertain* (U_j low). That indicates a confident-but-possibly-wrong shortcut. - We apply ASBN only to tokens DSCD flagged as ambiguous ($g_j = 1$) to focus compute. - Clipping $\lambda_{k,j}$ stabilizes training.

How (practical) - Compute $p_{\max}(j)$ as $\max_k p_{j,k}$. - Combine DSCD uncertainty signals into $U_j \in [0, 1]$ beforehand (entropy normalized + MC-var + σ-Net + novelty distance). - Implement clipping and optionally use a small smoothing (e.g., add 10^{-6}) to avoid zero division in other parts. - Set base values like $\bar{\lambda}_{\text{freq}} = 1.0$, $\bar{\lambda}_{\text{ctx}} = 0.5$, $\bar{\lambda}_{\text{xl}} = 0.8$, and $\lambda_{\max} = 2.0$ to start; tune later.

Step 2: Build discriminator inputs via GRL

What Create the input vectors for each discriminator by sending h_j through a GRL (with $\lambda_{k,j}$) and concatenating small side-features appropriate to each discriminator:

- D_{freq}: $\text{GRL}(h_j, \lambda_{\text{freq},j}) \oplus F_{w_j}$ (type-level priors).

- D_{ctx}: GRL$(h_j, \lambda_{\text{ctx},j}) \oplus$ context statistics $\oplus\, U_j$.
- D_{xl}: GRL$(h_j, \lambda_{\text{xl},j}) \oplus$ projected aligned target embedding.

Why Discriminators require both the representation (so they can learn to exploit shortcuts) and a minimal set of side information which signals the specific shortcut to watch for (e.g., frequency vector for D_{freq}). Using small, interpretable features prevents discriminators from overpowering the encoder with huge capacity.

How - **GRL:** use a GRL layer from your framework (PyTorch implementations exist) or implement manually: forward is identity, backward multiplies gradients by $-\lambda$. If per-sample λ support is unavailable, see the "per-sample GRL alternative" note below. - **Feature prep:** F_{w_j} can be a normalized histogram of sense counts or simply index-of-majority. ctx_stats_j can be sentence length, number of nearby content tokens, average attention mass, POS-based counts, etc. - **Projection:** apply a tiny MLP or linear projector to z^{tgt} before concatenation so dimensionalities align with discriminator input.

Step 3: Discriminator forward: compute per-token losses

What Run each discriminator on its inputs and compute a loss:

$$\begin{aligned} L_{\text{freq}} &= \frac{1}{N}\sum_j \text{CE}\big(D_{\text{freq}}(x_{\text{freq},j}), y_j^{\text{freq}}\big), \\ L_{\text{ctx}} &= \frac{1}{N}\sum_j \text{CE}\big(D_{\text{ctx}}(x_{\text{ctx},j}), y_j^{\text{ctx}}\big), \\ L_{\text{xl}} &= \frac{1}{N}\sum_j \text{ContrastiveOrCE}\big(D_{\text{xl}}(x_{\text{xl},j}),\, z^{\text{tgt}}_{a(j)},\, \text{neg}_j\big). \end{aligned}$$

Labels:

- y_j^{freq}: binary indicator if $\hat{y}_j$ equals the majority sense of F_{w_j}.
- y_j^{ctx}: binary indicator if context is sparse AND $\hat{y}_j$ matched majority (compute via heuristics).
- D_{xl} uses positive aligned target and negatives (wrong-sense target tokens drawn from prototypes).

Why Each loss trains a discriminator to detect a particular shortcut. Later, the encoder will receive reversed gradients via GRL which will force the encoder to reduce the discriminators' ability to detect these shortcuts.

How - Use small MLPs (1–2 layers) for discriminators. Keep them low capacity. - For D_{xl} a contrastive loss (InfoNCE) often works better: positives are real aligned target embeddings, negatives are plausible wrong-sense words (sample from prototype lexicon). - For y_j^{ctx} you can define context-sparse as sentence length $< L_{\text{thresh}}$ or DSCD entropy H_j above a threshold; choose a simple rule to avoid noise.

Step 4: Aggregate ASBN loss

What Combine the three discriminator losses into a single scalar:

$$L_{\text{ASBN}} = w_{\text{freq}} L_{\text{freq}} + w_{\text{ctx}} L_{\text{ctx}} + w_{\text{xl}} L_{\text{xl}}.$$

Why A single aggregated term keeps integration into the overall loss simple, allows weighting the importance of each adversarial signal, and is convenient for optimizer steps.

How Start with equal weights ($w_k = 1$) or if you know a particular shortcut is more harmful (e.g., frequency bias), give it more weight. Monitor training and adjust.

Step 5: Update discriminators (minimize L_{ASBN})

What Use optimizer step(s) to minimize L_{ASBN} w.r.t discriminator parameters ϕ:

$$\phi \leftarrow \phi - \eta_\phi \nabla_\phi L_{\text{ASBN}}.$$

Why Discriminators must be competent detectors, otherwise the encoder will have nothing meaningful to hide from. Keeping the discriminators up-to-date ensures the adversarial game remains informative.

How - Use a smaller learning rate for discriminators (η_ϕ) than the encoder typically. - Optionally perform multiple small discriminator steps per encoder step during early training (e.g., 2:1) to keep critics competitive.

Step 6: Compute primary model losses (DSCD + NMT)

What Compute the standard task losses coming from DSCD and decoder:

- L_{MT}: translation loss (cross-entropy of decoder outputs vs target).
- L_{span}: span detection loss (binary cross-entropy for ambiguous token flags).
- L_{sense}: supervised or contrastive sense loss (CE with gold/pseudo labels or contrastive clustering loss).

Why Adversarial training must not destroy task performance. These primary losses keep the model learning the main job: good translation and accurate span/sense predictions.

How - If gold sense labels exist for some tokens use supervised CE; otherwise use contrastive or clustering losses (as in DSCD) with prototypes. - Scale these losses using λ_{span} and λ_{sense} to balance training.

Step 7: Update encoder & decoder (GRL supplies reversed gradients)

What Compute total loss:

$$L_{\text{total}} = L_{\text{MT}} + \lambda_{\text{span}} L_{\text{span}} + \lambda_{\text{sense}} L_{\text{sense}} + \lambda_{\text{ASBN}} L_{\text{ASBN}},$$

then update encoder+DSCD+decoder parameters θ with gradient step:

$$\theta \leftarrow \theta - \eta_\theta \nabla_\theta L_{\text{total}}.$$

Why Including L_{ASBN} in the total loss ensures the encoder receives gradients that will *maximize* the discriminator objectives (because of the GRL) — equivalently, the encoder is being trained to hide shortcut signals while still optimizing the main tasks.

How - In practice GRL layers in the forward graph produce negated gradients automatically for the encoder path; ensure GRL is placed on the path from h_j into each discriminator. - If your framework does not support per-sample GRL scalars, one practical alternative is to multiply each per-token discriminator loss $\ell_{k,j}$ by $\lambda_{k,j}$ (stop-gradient on $\lambda_{k,j}$), average, and use that as L_k; this produces an identical encoder gradient sign/scale effect.

4 Implementation details & practical tips

1. **Warm-up:** Train DSCD + NMT (without strong ASBN) for 1–3 epochs so p_j and U_j are meaningful. Turn on ASBN after warm-up.
2. **Discriminator size:** use 1–2 layer MLPs with small hidden dims (32–256). Keep them weaker than encoder.
3. **Learning rates:** use lr_θ (encoder) $\approx$ 3e$-$4, lr_ϕ (discriminators) $\approx$ 1e$-$4 as starting points.
4. **Clipping λ values:** prevents reversed gradients from exploding. Clip to [0,2.0] initially.
5. **Alternate updates:** optionally run 1–3 discriminator steps per encoder step in early phases.
6. **Per-sample GRL alternative:** if framework lacks per-sample GRL, weight per-sample losses by $\lambda_{k,j}$ (stop-gradient on λ), average, then backward.
7. **Logging:** track discriminator accuracy (should drop over time), frequency-gap metric (majority vs minority sense accuracy), average $\lambda_{k,j}$, and homograph-specific BLEU/COMET.
8. **Debugging:** if translation quality drops, reduce λ_{ASBN} or $\bar{\lambda}_k$. If discriminators never degrade, increase GRL slowly.

5 Worked example (intuition)

Token: Bengali “” in sentence “ ” (expected sense: *page*).

- DSCD outputs: $p = (0.70\text{ leaf},\ 0.28\text{ page},\ 0.02\text{ blade})$, $p_{\max} = 0.70$, $U \approx 0.2$, $g = 1$.
- Compute $\lambda_{\text{freq}} \approx \bar{\lambda}_{\text{freq}} \cdot 0.7 \cdot (1 - 0.2) = \bar{\lambda}_{\text{freq}} \cdot 0.56$ (non-trivial reversed gradient).
- D_{freq} sees that chosen sense matches majority and minimizes L_{freq}. GRL causes encoder to receive reversed gradient making h_j more sensitive to context token , shifting p toward *page* in subsequent steps.

6 Hyperparameter suggestions (starting points)

- $\bar{\lambda}_{\text{freq}} = 1.0,\ \bar{\lambda}_{\text{ctx}} = 0.5,\ \bar{\lambda}_{\text{xl}} = 0.8,\ \lambda_{\max} = 2.0.$
- $w_k = 1.0$ (equal weights), $\lambda_{\text{ASBN}} = 0.2$ (global multiplier, tune upwards carefully).
- Discriminator hidden dim: 64; 1–2 layers with ReLU.
- Warm-up: 1–3 epochs for DSCD/NMT before full ASBN.

Per-sample GRL implementation note

Some frameworks do not support per-sample gradient scaling inside GRL. Equivalent practical option:

- Compute per-token discriminator losses $\ell_{k,j}$ normally (no GRL).
- Multiply each $\ell_{k,j}$ by $\lambda_{k,j}$ (stop-gradient on $\lambda_{k,j}$ so discriminators optimize the original loss, not scaled λ).
- Average to get $L_k = \frac{1}{N}\sum_j \lambda_{k,j}\ell_{k,j}$.
- Backpropagate L_{ASBN} as usual. The encoder will receive gradients scaled by $\lambda_{k,j}$ and with opposite sign if you insert a GRL-like manual sign for encoder update (or you can implement encoder update by subtracting those gradients manually). In many practical cases this weighted-loss trick is simpler and yields the intended effect.

Algorithm 2 ASBN Training Step (Descriptive Pseudocode)

Require: Encoder outputs (token representations), token metadata (confidence, uncertainty, frequency, context, alignment)

Step 1: Compute GRL strengths
- **What:** Assign importance weights for each token and discriminator.
- **Why:** Apply stronger adversarial pressure only when bias is likely.
- **How:** Use model confidence, uncertainty, and ambiguity flag.

Step 2: Build discriminator inputs
- **What:** Prepare features for each discriminator.
- **Why:** Each discriminator detects a different shortcut bias.
- **How:**
  - Frequency: encoder representation + frequency stats.
  - Context: encoder representation + context statistics.
  - Cross-lingual: encoder representation + aligned target embedding.

Step 3: Run discriminators and compute losses
- **What:** Train discriminators to detect shortcuts.
- **Why:** Encourage encoder to hide shortcut signals.
- **How:** Forward pass inputs, compare with labels, compute losses.

Step 4: Aggregate ASBN loss
- **What:** Combine discriminator losses.
- **Why:** Balance multiple shortcut detectors.
- **How:** Weighted sum of frequency, context, and cross-lingual losses.

Step 5: Update discriminators
- **What:** Optimize discriminators to improve shortcut detection.
- **Why:** Keep adversarial game strong.
- **How:** Perform gradient descent minimizing ASBN loss (update only discriminator parameters).

Step 6: Compute main model losses
- **What:** Calculate translation, span, and sense losses.
- **Why:** Maintain task-specific objectives as the core training signal.
- **How:** Use standard NMT and DSCD loss functions.

Step 7: Update encoder and decoder
- **What:** Train encoder and decoder with task + adversarial objectives.
- **Why:** Make encoder sense-aware while reducing shortcut reliance.
- **How:** Combine all losses, apply GRL to reverse discriminator gradients, update model parameters.

Ensure: Updated encoder & decoder (less biased, more sense-aware), and trained discriminators.