# Homograph Disambiguation in NMT for Low-Resource Languages (TATN — Tri-Modal Adversarial Transparency Network)

## 1 Topic: title and description

**Title:** *Homograph Disambiguation in Neural Machine Translation for Low-Resource Languages.*

**Brief description** The research investigates methods to detect and resolve homographs (words with multiple distinct senses) in neural machine translation (NMT) for low-resource language pairs (example: Bengali → English). The proposed solution augments an encoder–decoder NMT with three cooperating modules:
**DSCD** (to detect ambiguous tokens and maintain online sense prototypes),
**ASBN** (to adversarially remove shortcut/spurious cues so the model relies on real contextual evidence), and
**TRG** (to generate and verify human-readable rationales for sense choices). Together these modules improve disambiguation accuracy and produce faithful explanations for decisions, which is especially valuable in low-resource settings where model overfitting to frequency is common.

## 2 Three main components of TATN

TATN integrates three modules. Below each module is described with its subparts, formulas, purpose, and interaction with other modules.

### 2.1 DSCD — Dynamic Span–Sense Co-Detection

**Purpose.** Detect tokens or spans needing disambiguation, estimate their sense distribution, maintain online sense prototypes, and output multi-source uncertainty that drives both ASBN (adversarial strength) and TRG (rationale extraction).

**High-level steps / parts**

- **Buffers and prototype store**: per-surface-type circular buffer $B_w$ and prototype list $C_w = \{c_{w,1}, \ldots, c_{w,K_w}\}$. Buffers collect recent encoder vectors for type $w$; prototypes are centroids representing learned senses.
- **Type-level dispersion** $D_w$: measure of contextual variance across $B_w$. If $D_w > \delta_{\text{type}}$ mark type as candidate for multi-sense handling.

$$\bar{h}_w = \frac{1}{|B_w|} \sum_{x \in B_w} x, \qquad D_w = \frac{1}{|B_w|} \sum_{x \in B_w} \|x - \bar{h}_w\|^2.$$

- **Per-token sense distribution**: compute cosine similarity to prototypes and softmax with temperature $T$:

$$s_{j,i} = \frac{h_j^\top c_{w,i}}{\|h_j\|\|c_{w,i}\|}, \qquad p_{j,i} = \frac{\exp(s_{j,i}/T)}{\sum_k \exp(s_{j,k}/T)}.$$

- **Predictive uncertainty (multi-source)**:
  - *Entropy* $H_j = -\sum_i p_{j,i} \log(p_{j,i} + \epsilon)$.
  - *MC-dropout variance* $\mathrm{Var}_j$ from $M$ stochastic passes.
  - *Aleatoric / $\sigma$-Net* $\sigma_j = \exp(\frac{1}{2}u_j)$ where $u_j = W_\sigma h_j + b_\sigma$.
  - *Novelty* $d_{\min} = \min_i(1 - s_{j,i})$.

  These combine into a single scalar $U_j$ (additive default, or normalized/weighted sigmoid form for fine control).
- **Dynamic clustering / prototype maintenance**:
  - Track rolling assignment distances (mean $\mu_w$, std $\tau_w$).
  - Adaptive creation threshold $\varepsilon_{\text{new},w} = \mu_w + \lambda\tau_w$.
  - If $d_{\min} > \varepsilon_{\text{new},w}$ (and optionally $U_j > \delta_{\text{inst}}$) create new prototype $c_{w,K_w+1} \leftarrow h_j$ (cap with $K_{\max}$), else assign to nearest and update centroid via EMA:

$$c_{w,i^\star} \leftarrow (1 - \eta)c_{w,i^\star} + \eta h_j.$$

  - Optionally require $N_{\min}$ assignments before a prototype is stable.
- **Span detector and gating**: span head $\hat{b}_j = \sigma(W_{\text{span}} h_j + b_{\text{span}})$ and an attention gate $g_j = \sigma(w_g(U_j - b_g))$; the gate controls attention boosting and ASBN strength.
- **Sense-augmented embedding**: for flagged tokens, augment encoder vector with centroid:

$$h'_j = h_j + c_{w,\hat{y}_j}.$$

**Why DSCD is important** DSCD provides explicit sense signals and a numeric uncertainty estimate that tells the system *which* tokens need special handling. Prototypes act as interpretable, stored senses; uncertainty prevents overconfident wrong edits and guides TRG and ASBN in a targeted way. DSCD is particularly useful for low-resource languages because it builds sense anchors online and does not require large annotated sense inventories.

## 2.2 ASBN — Adversarial Sense-Balance Network

**Purpose.** Prevent the model from exploiting spurious shortcuts (frequency, shallow context patterns, or alignment leakage) to resolve homographs. It does so with token-level adversarial discriminators applied through Gradient Reversal Layers (GRL) with per-token strengths determined by DSCD signals.

**High-level parts**

- **Discriminators** — specialized auditors:
  - $D_{\text{freq}}$: detects frequency-based shortcuts (i.e., predicts the high-frequency surface→sense mapping).
  - $D_{\text{ctx}}$: detects weak-context cues.
  - $D_{\text{xl}}$: detects cross-lingual alignment leakage or target-side cues.
- **Per-token GRL strength** (key innovation):

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

Intuition: stronger adversarial pressure where the model is confident ($p_{\max}$ high) and the token is important (gate $g_j$), but reduce pressure where uncertainty $U_j$ is high.

- **Discriminator inputs** are formed by concatenating GRL-transformed encoder vector and side features for each auditor:

$$x_{\text{freq}} = \text{concat}(\text{GRL}(h_j, \lambda_{\text{freq},j}), F_w),$$

$$x_{\text{ctx}} = \text{concat}(\text{GRL}(h_j, \lambda_{\text{ctx},j}), \text{ctx\_stats}_j, U_j),$$

$$x_{\text{xl}} = \text{concat}(\text{GRL}(h_j, \lambda_{\text{xl},j}), \text{proj}(z_{\text{tgt\_attn},j})).$$

- **Losses** and aggregation:

$$L_{\text{freq}} = \frac{1}{|B|} \sum_j \text{CE}(D_{\text{freq}}(x_{\text{freq},j}), y_j^{(\text{freq})}), \text{ etc.}$$

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

- **Update order**: discriminators $\phi$ are updated *first* to minimize $L_{\text{ASBN}}$ (small LR); encoder parameters $\theta$ are updated after, with GRL causing the encoder to maximize (i.e., confuse) the discriminators. This order is crucial for stable adversarial training.

**Why ASBN is important** ASBN forces the encoder to stop relying on spurious cues. Especially in low-resource settings, frequency biases and shallow heuristics dominate; adversarial removal of these shortcuts leads to representations that must use true contextual cues and prototypes, thus improving homograph disambiguation robustness.

### 2.3 TRG — Transparent Rationale Generator

**Purpose.** Produce human-readable, faithful rationales explaining the sense decision for ambiguous tokens, and enforce faithfulness via a verifier. TRG both improves interpretability and serves as a training signal (via verifier feedback).

**High-level parts**

- **Extractor**: for each flagged token, gathers structured evidence $E_j$:
  - local attention context,
  - nearest prototypes and distances,
  - ASBN diagnostics (which discriminators flagged what),
  - alternative candidate senses and confidence scores.
- **Formatter / Template fallback**: deterministic, slot-filling rationales (e.g., "Because the context contains X and prototype Y, the sense is Z") used as silver labels to bootstrap generator training.
- **Generator** $G$: seq2seq (small) that maps structured input $X_j$ to fluent rationale $R_j$. Trained on silver set $(X_j, R_j^T)$ where templates provide $R_j^T$.
- **Verifier** $V$: classifier that, given $(S \oplus R)$, predicts the sense. Used to accept/reject generated rationales (ensures they are faithful).
- **Losses**: generator CE loss $L_{\text{gen}}$, fidelity loss $L_{\text{fid}}$ (verifier CE), coverage loss $L_{\text{cov}}$ encouraging inclusion of required evidence.

$$L_{\text{TRG}} = L_{\text{gen}} + \lambda_{\text{fid}} L_{\text{fid}} + \lambda_{\text{cov}} L_{\text{cov}}.$$

- **Inference-time policy**: For flagged tokens: if $p_{\max} > \tau_{\text{high}}$ and $U_j < \tau_{\text{low}}$ use template; otherwise generate candidate rationales with $G$, and accept a candidate only if $V(S \oplus r)$ predicts the same sense with confidence $> \tau_{\text{accept}}$. If none pass, fall back to template.

**Why TRG is important** TRG enforces faithfulness and provides human-understandable explanations. In training it also helps the model by making sense decisions explicit and verifiable. The verifier loops the explanation back into a check for sense consistency.

# 3 Why two algorithms (Phase A and Phase B)?

TATN is presented as two algorithms because training and inference are different operations with distinct computational and algorithmic needs:

- **Phase A (Training)** requires gradient computations, parameter updates, discriminator training (with GRL), prototype creation/updates, and generator/verifier training. It includes $\nabla_\phi L_{\text{ASBN}}$ and $\nabla_\theta L_{\text{total}}$ steps, which do not occur at inference. Moreover, training includes methods that create or adjust model-internal data structures (prototypes) in an online fashion.
- **Phase B (Inference)** is a purely forward pass: compute DSCD outputs, run discriminators only in diagnostic mode (no parameter updates), compute $h'_j$ and boosted attention, decode translation, and optionally run TRG generation + verifier acceptance. Prototype creation is typically disabled or tightly controlled at inference (unless continual learning is desired).

Separation clarifies responsibilities, reduces ambiguity (easily seen when reading pseudocode), and prevents dangerous mixing of parameter updates and production-time behavior (e.g., accidental prototype creation during translation in a deployed system).

# 4 Revised TATN algorithm (pseudocode)

Below is a self-contained pseudocode (phase A + phase B)

**Algorithm 1** TATN — Phase A (training) and Phase B (inference) — compact pseudocode

**Require:** pretrained Encoder, Decoder; SentencePiece/BPE; empty prototype stores $C_w$ and circular buffers $B_w$; hyperparameters

**Phase A: Training**
**for** each minibatch of source sentences $S$ and targets $T$ **do**
  **for** each token $s_j \in S$ **do**
    $h_j \leftarrow \text{Encoder}(s_j)$; append $h_j$ to $B_{w_j}$
  **end for**
  **for** each type $w$ with updated buffer **do**
    compute $\bar{h}_w$ and $D_w$; if $D_w > \delta_{\text{type}}$ mark type candidate
  **end for**
  **for** each token $j$ **do**
    **if** $C_{w_j} = \varnothing$ **then**
      seed proto (append to seed buffer until $N_{\min}$ then $c_{w,1} \leftarrow$mean)
    **else**
      compute $s_{j,i}$ for $i \in [1..K_w]$; $p_j \leftarrow \text{softmax}(s_j/T)$; $\hat{y}_j \leftarrow \arg\max p_j$
      compute $H_j$, $d_{\min,j}$, MC dropout $\text{Var}_j$, $\sigma_j$ (via $\sigma$-net)
      aggregate $U_j \leftarrow H_j + \text{Var}_j + \sigma_j + d_{\min,j}$ (or normalized-weighted form)
      if $d_{\min,j} > \varepsilon_{\text{new},w}$ and $U_j > \delta_{\text{inst}}$ then create prototype else EMA-update nearest centroid
    **end if**
    compute span probability $\hat{b}_j = \sigma(W_{\text{span}} h_j + b_{\text{span}})$
    compute gate $g_j = \sigma(w_g(U_j - b_g))$ and attention boost $\tilde{a}_j \leftarrow a_j^{(0)}(1 + \gamma g_j)$
    **if** flagged (type candidate and $U_j > \delta_{\text{inst}}$) or $\hat{b}_j > 0.5$ **then**
      $h'_j \leftarrow h_j + c_{w_j,\hat{y}_j}$
    **else**
      $h'_j \leftarrow h_j$
    **end if**
  **end for**
  **ASBN:** for each discriminator $k \in \{\text{freq,ctx,xl}\}$ and token $j$ compute

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

  and build inputs $x_{k,j} = \text{concat}(\text{GRL}(h_j, \lambda_{k,j}), \text{side\_features})$
  compute discriminator losses $L_{\text{freq}}, L_{\text{ctx}}, L_{\text{xl}}$ and aggregate $L_{\text{ASBN}}$
  **Update discriminator params** $\phi \leftarrow \phi - \eta_\phi \nabla_\phi L_{\text{ASBN}}$
  compute translation loss $L_{MT}$, span loss $L_{\text{span}}$, sense loss $L_{\text{sense}}$
  $L_{\text{total}} \leftarrow L_{MT} + \lambda_{\text{span}} L_{\text{span}} + \lambda_{\text{sense}} L_{\text{sense}} + \lambda_{\text{ASBN}} L_{\text{ASBN}} + \lambda_{\text{reg}} R(\theta)$
  **Update encoder/decoder params** $\theta \leftarrow \theta - \eta_\theta \nabla_\theta L_{\text{total}}$ (GRL reverses ASBN gradients)
  **TRG silver collection:** for flagged tokens with $U_j > \tau_U$ extract evidence $E_j$, build $X_j$, create template $R_j^T$, add $(X_j, R_j^T, \hat{y}_j)$ to TRG dataset
  update Generator $G$ and Verifier $V$ with their joint loss $L_{\text{TRG}} = L_{\text{gen}} + \lambda_{\text{fid}} L_{\text{fid}} + \lambda_{\text{cov}} L_{\text{cov}}$
**end for**

**Phase B: Inference**
**for** each input sentence $S$ **do**
  **for** each token $s_j$ **do**
    $h_j \leftarrow \text{Encoder}(s_j)$; compute $s_{j,i}, p_j, H_j, d_{\text{min},j}, \text{Var}_j, \sigma_j, U_j, g_j, \hat{b}_j$
    set $h'_j \leftarrow h_j + c_{w_j,\hat{y}_j}$ if flagged / $\hat{b}_j > 0.5$ else $h'_j \leftarrow h_j$
  **end for**
  decode using $\{h'_j\}$ and $\{\tilde{a}_j\}$ to produce $\hat{T}$
  **Rationale generation:** for each flagged token:
  **if** $p_{\max} > \tau_{\text{high}}$ and $U_j < \tau_{\text{low}}$ **then** output template rationale
  **else**generate candidate rationales $\{r_c\} = G(X_j)$ and accept $r_c$ if $V(S \oplus r_c)$ predicts $\hat{y}_j$ with confidence $> \tau_{\text{accept}}$; else fallback to template
  **end if**
  output translation + per-token sense and rationale bundle
**end for**

# 5 Line-by-line explanation of the algorithm (module boundaries shown)

Below we present the Phase A (training) and Phase B (inference) explanations, with explicit module boundary markers indicating where each module **starts** and **ends**.

## 5.1 Phase A — Training (step-by-step with module boundaries)

**Module boundary summary (Phase A):**

- **DSCD:** starts at **Step 2** and ends at **Step 12**.
- **ASBN:** starts at **Step 13** and ends at **Step 18**.
- **TRG:** starts at **Step 19** and ends at **Step 20**.
- **Logging/monitoring:** Step 21 (shared/afterwards).

1. **Initialization (global)**: tokenizer, encoder/decoder, prototype stores $C_w$, circular buffers $B_w$, hyperparameters. (Prepares environment for all modules.)
2. **[DSCD] — Encode & buffer append:** For each token $s_j$, compute $h_j = \text{Encoder}(s_j)$ and append $h_j$ to $B_{w_j}$. This begins DSCD: collecting per-type context vectors for subsequent dispersion/prototype work.
3. **Type-level dispersion $D_w$:** Compute $\bar{h}_w$ and $D_w = \frac{1}{|B_w|}\sum \|x - \bar{h}_w\|^2$. If $D_w > \delta_{\text{type}}$ mark the type $w$ as a multi-sense candidate. (DSCD decides whether heavy sense-processing is needed.)
4. **Prototype seeding (if $C_w = \varnothing$):** Option A immediate create or Option B buffered creation after $N_{\min}$ examples. (DSCD chooses a stable initialization policy.)
5. **Similarity & sense posterior:** Compute cosine sims $s_{j,i}$ to prototypes and $p_j = \text{softmax}(s_j/T)$; set $\hat{y}_j = \arg\max p_j$. (DSCD computes candidate senses.)
6. **Entropy and novelty:** Compute entropy $H_j$ and novelty $d_{\text{min},j}$. These are DSCD signals indicating ambiguity and out-of-distributionness.
7. **Epistemic (MC-dropout):** Run $M$ stochastic passes to obtain $\text{Var}_j$. (Part of DSCD's multi-source uncertainty.)
8. **Aleatoric ($\sigma$-Net):** Predict $u_j = W_\sigma h_j + b_\sigma$ and $\sigma_j = \exp(\frac{1}{2}u_j)$. (DSCD learns per-token noise.)

9. **Aggregate uncertainty $U_j$:** Form $U_j$ from $H_j, \mathrm{Var}_j, \sigma_j, d_{\min,j}$ (additive default or normalized-weighted sigmoid). This scalar controls gating, prototype creation, and ASBN strength.
10. **Dynamic clustering (create/assign/update prototypes):** Using rolling stats $(\mu_w, \tau_w)$ and threshold $\varepsilon_{\text{new},w} = \mu_w + \lambda\tau_w$, either create new prototype $c_{w,K+1} \leftarrow h_j$ or EMA-update existing centroid $c_{w,i^\star} \leftarrow (1-\eta)c_{w,i^\star} + \eta h_j$. Optionally require $N_{\min}$ assignments before a prototype is stable.
11. **Span detection & gating:** Compute span score $\hat{b}_j = \sigma(W_{\text{span}} h_j + b_{\text{span}})$ and gate $g_j = \sigma(w_g(U_j - b_g))$. If flagged, mark token for special handling (attention boost, TRG silver extraction).
12. **Sense-augmented embedding:** For flagged tokens (or if $\hat{b}_j > 0.5$) set $h'_j = h_j + c_{w,\hat{y}_j}$; otherwise $h'_j = h_j$. DSCD ends here for this batch: it has produced $p_j, U_j, g_j, h'_j$ used by ASBN and TRG.
13. **[ASBN]Per-token GRL strengths:** For each discriminator $k \in \{\text{freq,ctx,xl}\}$ compute

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

    This is the ASBN hook: DSCD outputs (especially $p_{\max}, U_j, g_j$) directly parameterize ASBN behavior.
14. **Build discriminator inputs with GRL:** Form $x_{k,j} = \text{concat}(\text{GRL}(h_j, \lambda_{k,j}), \text{side\_features}_k)$ (e.g., frequency priors, context stats, projected target-attn). Feed to discriminators $D_k$.
15. **Discriminator forward & losses:** Compute $L_k$ for each auditor (CE or contrastive as appropriate) and aggregate $L_{\text{ASBN}} = \sum_k w_k L_k$. These losses train discriminators to detect shortcuts.
16. **Update discriminators:** Update discriminator params $\phi$ to minimize $L_{\text{ASBN}}$ (use smaller LR, optional multi-step). This step must precede encoder update so discriminators are competent.
17. **Compute primary model losses:** Compute $L_{MT}$, $L_{\text{span}}$, and $L_{\text{sense}}$. (ASBN has prepared $L_{\text{ASBN}}$ for inclusion.)
18. **Update encoder/decoder with reversed gradients:** Build total loss

$$L_{\text{total}} = L_{MT} + \lambda_{\text{span}} L_{\text{span}} + \lambda_{\text{sense}} L_{\text{sense}} + \lambda_{\text{ASBN}} L_{\text{ASBN}} + \lambda_{\text{reg}} R(\theta),$$

    then update encoder/decoder params $\theta$ (GRL ensures the encoder experiences reversed ASBN gradients and thus learns to hide shortcuts). ASBN ends after the encoder update for this batch.
19. **[TRG]Silver rationale extraction:** For tokens flagged by DSCD (and with $U_j > \tau_U$) collect evidence $E_j$ (attention context, prototypes, ASBN diagnostics, alternatives), form structured input $X_j$, and create deterministic template rationale $R_j^T$. Add $(X_j, R_j^T, \hat{y}_j)$ to TRG silver dataset. This is the start of TRG operations.
20. **Train Generator & Verifier:** Use silver dataset to train generator $G$ (minimize $L_{\text{gen}}$) and verifier $V$ (minimize $L_{\text{fid}}$), and include coverage loss $L_{\text{cov}}$ so that the generator mentions required evidence. Update TRG params $(\psi_G, \psi_V)$ via $L_{\text{TRG}} = L_{\text{gen}} + \lambda_{\text{fid}} L_{\text{fid}} + \lambda_{\text{cov}} L_{\text{cov}}$.
21. **Finalize batch updates & logging:** TRG finishes its updates. Log prototype stats, U histograms, discriminator accuracy, TRG acceptance rate etc. (Step 21 is post-update monitoring shared across modules.)

### 5.2 Phase B — Inference (module boundaries shown)

**Module boundary summary (Phase B):**

- **DSCD:** runs in forward-only mode (computes $p_j, U_j, g_j, h'_j$). DSCD operations correspond to Phase A DSCD lines (no prototype creation unless allowed).
- **ASBN:** discriminators are *not* trained; they can be evaluated for diagnostics, but no GRL/updates occur. ASBN's role in inference is diagnostic only.
- **TRG:** runs generator/verifier in forward inference to produce and validate rationales.

1. [**DSCD START - Inference**] — **Forward DSCD computations:** For each token compute $h_j$, $s_{j,i}$, $p_j$, $H_j$, $\mathrm{Var}_j$, $\sigma_j$, $d_{\min,j}$, aggregate $U_j$, compute $g_j$ and $\hat{b}_j$; form $h'_j$ for flagged tokens. (DSCD performs same computations as Phase A but *without* parameter updates or prototype creation unless explicitly enabled.)
2. [**ASBN - Inference**] — **Diagnostics (optional):** Run discriminators on GRL(h) inputs for monitoring/diagnosis. *Do not* perform GRL updates or parameter changes. ASBN contributes only audit information at inference.
3. [**TRG START - Inference**] — **Rationale generation & verification:** For each flagged token:
   - If $p_{\max} > \tau_{\text{high}}$ and $U_j < \tau_{\text{low}}$ output template rationale.
   - Else generate candidates $\{r_c\} = G(X_j)$ and accept $r_c$ if $V(S \oplus r_c)$ predicts $\hat{y}_j$ with confidence $> \tau_{\text{accept}}$; otherwise fallback to template.

   TRG ends after candidate acceptance / fallback.
4. **Finalize inference output:** Decoder output (translation) together with per-token sense decisions and accepted rationales are returned to the user.

## 6 TikZ diagram of the algorithm

Below is a TikZ diagram that visualizes the dataflow: encode → DSCD → ASBN → TRG → decoding. Paste this into Overleaf; it is included in the document.

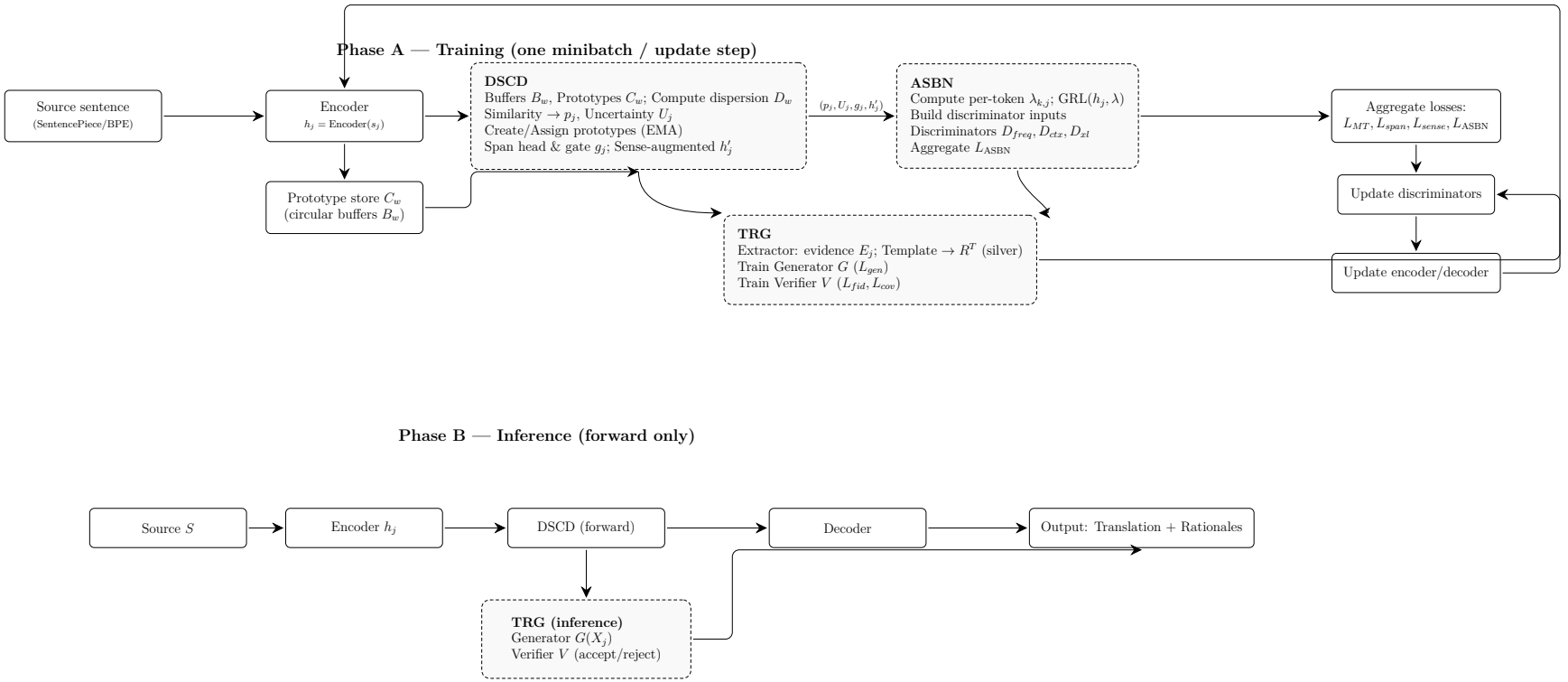


Figure 1: Landscape-mode TATN diagram with larger boxes and font. Arrows are routed via top and right lanes to avoid overlap and terminate exactly at their target blocks.

# 7 Implementation checklist (practical)

- **Data preprocessing:** SentencePiece/BPE model; token-level mapping to surface types $w$; optional morphological analyzers for languages like Bengali.
- **Model components:**
  - Encoder/Decoder (Transformer-based recommended).
  - DSCD: circular buffers $B_w$, prototype store $C_w$, $\sigma$-Net (MLP), span head.
  - ASBN: small MLP discriminators, GRL wrapper with per-token strength.
  - TRG: small seq2seq generator $G$ and verifier classifier $V$.
- **Key hyperparameters:** $T \in [0.6, 1.0]$, $M = 3$–$8$, $\eta \approx 0.03$–$0.08$, $N_{\min} = 3$–$10$, $K_{\max} \approx 20$, $\bar{\lambda}_{\text{freq}} = 1.0$, $\bar{\lambda}_{\text{ctx}} = 0.5$, $\bar{\lambda}_{\text{xl}} = 0.8$, $\lambda_{\max} \approx 1.0$–$2.0$.
- **Training schedule:** optional DSCD warmup 1-3 epochs; enable ASBN afterwards; TRG silver collection runs each epoch (or each few batches).
- **Monitoring:** prototype count  creation rate, discriminator training accuracy (should be non-trivial), U histograms, TRG acceptance rate, BLEU/COMET on flagged subsets.

# 8 Conclusion

TATN integrates DSCD, ASBN, and TRG to address homograph disambiguation by (1) explicitly modeling candidate senses and uncertainty, (2) removing shortcut signals through targeted adversarial training, and (3) producing verified rationales to make disambiguations transparent and to bootstrap further learning. For low-resource languages (e.g., Bengali $\rightarrow$ English) the modular design is especially valuable: DSCD builds online sense anchors, ASBN prevents frequency-driven errors, and TRG provides human-interpretable explanations that can be inspected or used for weak supervision.