ACSender: A Swarm-Inspired Transformer Architecture

Abstract

본 논문에서는 군집 지능 (Swarm Intelligence) 에서 영감을 받은 구조적 편향을 도입한 새로운 Transformer 아키텍처인 ASCender 를 제안한다. 기존 Self-Attention 메커니즘은 모든 토큰 쌍을 동일하게 처리함으로써 계산 효율성이 떨어지고, 결과 해석 가능성이 제한되는 한계를 가진다. 이러한 균일한 처리 방식은 특히 계층적 구조나 공간적 추론이 필요한 과제에서 도메인 특유의 관계 구조를 충분히 반영하지 못한다.

ASCender 는 집단 행동 모델인 Boids 에서 유래한 세 가지 유도 편향 (Alignment, Separation, Cohesion) 을 Attention Score 계산 과정에 직접 통합한다. 이를 통해 의미적으로 유사한 토큰을 정렬 (Alignment) 하고, 불필요하거나 잡음이 많은 토큰과는 분리 (Separation) 하며, 의미 있는 군집 내로 응집 (Cohesion) 시키는 과정을 유도한다. 이러한 Swarm-aware Attention 설계는 해석 가능한 Attention Map 을 생성하고, 불필요한 토큰 상호작용을 억제하며, 장문 컨텍스트 처리 효율성을 향상시킨다.

다양한 자연어 처리 및 추론 벤치마크 실험에서 ASCender 는 기존 Transformer 대비 최대 8.3% 의 정확도 향상과 21% 의 Attention FLOPs 절감을 달성하였다. 이 연구 결과는 생물학적 영감을 받은 유도 편향이 신경망 Attention 메커니즘의 효율성과 해석 가능성을 동시에 개선할 수 있는 잠재력을 보여주며, 향후 군집 지능 원리를 딥러닝 아키텍처 설계에 접목할 가능성을 제시한다.

Introduction

Transformer 는 자연어 처리, 컴퓨터 비전, 멀티모달 학습 전반에서 혁신을 일으키며, 다양한 과제에서 사실상의 표준 아키텍처로 자리잡았다. Transformer 의 핵심인 Self-Attention 메커니즘은 시퀀스 내 모든 토큰 쌍의 상호작용을 계산한다. 그러나 이러한 메커니즘은 모든 토큰 쌍을 균일하게 처리하며, 명시적인 구조적 가이던스 없이 학습된 Attention 가중치에만 의존한다. 이로 인해 불필요하거나 중복된 토큰 간 상호작용에 계산 자원이 소모되고, Attention 패턴이 해석하기 어렵게 나타나는 문제가 발생한다.

최근 연구에서는 Sparse Attention, 저랭크 근사 (Low-rank Approximation), Positional Encoding 개선 등으로 이러한 한계를 보완하려 하였다. 이러한 방법들은 계산 효율성이나 컨텍스트 모델링을 개선할 수 있지만, 특정 도메인에서 토큰이 서로 어떻게 관계 맺어야 하는지에 대한 명시적인 유도 편향 (Inductive Bias) 을 포함하지 않는 경우가 많다. 적절히 설계된 유도 편향은 학습을 보다 의미 있고 해석 가능한 관계 구조로 이끌 수 있다.

본 연구에서는 군집 지능 (Swarm Intelligence), 특히 집단 행동 모델인 Boids 에서 영감을 받아 Transformer 에 적용 가능한 새로운 형태의 구조적 편향을 설

계하였다. Boids 모델은 단순하지만 강력한 세 가지 규칙을 통해 집단의 거동을 설명한다: Alignment(이웃의 평균 방향으로 정렬), Separation(과도한 밀집 회피), Cohesion(이웃의 평균 위치로 응집). 우리는 이 원리를 Self-Attention 맥락에서 재해석하여 Attention Score 계산 과정에 직접 반영하였다.

이를 위해 Alignment–Separation–Cohesion–enhanced Transformer 인 **ASCender** 를 제안한다. ASCender 는 의미적으로 유사한 토큰 군집을 형성하도록 유도하고, 불필요한 상호작용을 줄이며, 장문 컨텍스트 처리 효율을 향상시킨다. NLP 와 추론 벤치마크 실험을 통해 ASCender 가 기존 Transformer 보다 더 높은 정확도와 해석 가능성을 달성함과 동시에 계산 효율도 개선함을 보인다.

본 논문의 기여 사항은 다음과 같다. 1. Alignment, Separation, Cohesion 원리를 구조적 편향 형태로 Self-Attention 에 통합하는 새로운 접근법 제안 2. 편향을 Attention Score 행렬에 직접 삽입하여 해석 가능한 Attention 패턴과 효율적인 토큰 상호작용 구현 3. 다양한 벤치마크 실험을 통한 성능 및 계산 효율성 향상 검증

Related Work

1) 효율화 지향 Attention (희소화, 저랭크, 커널, 메모리)

Self-Attention 의 **계산 복잡도**를 낮추는 연구들입니다. 슬라이딩 윈도·블록·클러스터/라우팅·글로벌 토큰·랜덤 링크 등으로 밀도를 줄이고, 저랭크/커널 기법으로 선형화하며, 커널 융합으로 실제 시간과 메모리를 절감합니다. **ASCender 와의 관계.** 이들은“얼마나 덜 계산할 것인가”에 집중합니다. ASCender 는“**무엇을 우선 연결할 것인가**”를 바꾸므로 **직교적**이며, 서로 결합 가능합니다.

2) 위치·거리 편향

절대/상대/회전형 위치 인코딩과 선형 거리 패널티는 길이 일반화와 문서 단위 컨텍스트에 효과적입니다. **한계.** 이는 **기하학 중심**으로, 토큰 간 **의미적 상호작용 방향성**은 규정하지 않습니다. **ASCender.** 의미 유사 이웃과 **정렬·응집**, 과잉/잡음 이웃과 **분리**를 유도하는 **내용 기반** 편향을 추가합니다.

3) 유도 편향 (Inductive Bias)

구문/그래프 기반 엣지, 알고리즘/산술 일반화를 돕는 규제 등 **구조적 힌트**를 직접 주입하는 연구가 있습니다. 데이터 효율과 OOD 일반화가 개선됨이 보고됩니다. **ASCender 관점.** 고정 구조 (예: 구문트리) 주입과 달리, ASCender 는 ** 동역학적 연속 규칙 (A/S/C)** 을 사용해 **상황 적응적**으로 군집과 억제를 유도합니다.

4) 군집화·그룹핑·라우팅

주의를 **그룹 단위**로 조직하거나, 학습된 라우팅/클러스터링을 통해 비용을 줄이는 방법들, 그리고 **자발적 군집**이 드러난다는 분석 연구가 존재합니다. **차이점.** 기존

은 군집을 관찰/활용하는 경향이 강한 반면, ASCender 는 **명시적 편향**으로 **군집을 생성·유도**합니다.

5) 생물·스웜 영감

스웜 지능 (Boids, PSO 등) 은 최적화나 제어에서 널리 쓰였으나, **주의 점수 수준에 Boids 규칙을 직접 삽입**하는 접근은 드뭅니다. **ASCender.** Alignment/Separation/Cohesion 을 **가산 편향 항**으로 공식화하여, **에이전트 동역학 → 토큰 상호작용**의 직접적 매핑을 제시합니다.

6) 주의 해석 가능성

주의가 곧 설명은 아니라는 비판 이후, 안정화·분해 기법이 제안되었습니다. **ASCender.** 주의 점수를 **(기본) + (정렬) + (분리) + (응집)** 으로 분해해 **명명된 기여도**를 시각화할 수 있어 해석 가능성을 높입니다.

7) 장문 컨텍스트

긴 문맥은 위치 체계, 재귀/메모리, 희소 패턴의 결합으로 해결해왔습니다. 이는 도달성·비용 문제를 줄이지만, **관계 선택성**은 별도 과제입니다. **ASCender.** Separation 으로 **불필요 원거리 상호작용 억제**, Cohesion 으로 **핵심 군집 강화**를 유도해 장문 처리와 상호 보완적입니다.

정리 및 위치 지정

기존 연구는 (i) **속도**, (ii) **기하학**, (iii) **고정 구조 주입**, (iv) **사후 해석**에 집중해 왔습니다. ASCender 는 **연속·내용 기반·동역학적 유도 편향**을 통해

- **의미 군집을 형성** (정렬/응집),
- **중복·잡음을 억제** (분리),
- 희소·선형·플래시·장문 메모리와 **호환**되는 길을 제시합니다.

Method

1. Notation & Setup (영문/국문)

- Tokens $x_1, \dots, x_n \in \mathbb{R}^{d_{\text{model}}}$.
- Linear maps: $Q = XW_Q$, $K = XW_K$, $V = XW_V$ with $W_\bullet \in \mathbb{R}^{d_{\text{model}} \times d}$.
- Base attention scores: $S_{ij}^{\text{base}} = \frac{q_i^\top k_j}{\sqrt{d}}$.
- Let softmax_j denote row-wise softmax over index j.
- Optional positional inputs π_i (absolute/relative/rotary).
- Multi-head index h omitted where clear; all terms are per head unless stated.

(국문) 토큰 x_i 로부터 Q, K, V 를 얻고, 기본 점수 S^{base} 는 표준 점수입니다. 이후 제안하는 편향 항 $\beta_{\text{align}}, \beta_{\text{sep}}, \beta_{\text{coh}}$ 을 **softmax 이전**에 가산합니다.

$$S_{ij} = S_{ij}^{\text{base}} + \beta_{ij}^{\text{align}} + \beta_{ij}^{\text{sep}} + \beta_{ij}^{\text{coh}} + \beta_{ij}^{(\text{other})}, \quad A_{ij} = \text{softmax}_j(S_{ij}), \quad \text{Attn}(X) = AV.$$

여기서 $\beta^{(\text{other})}$ 는 ALiBi, 상대 위치 바이어스 등 기존 항과의 병용을 허용합니다.

---

2. Learned Latent Geometry (잠재 기하)

We introduce a **learned metric space** for grouping:

- Project to latent coordinates $z_i = Ux_i \in \mathbb{R}^{d_z}$ with $U \in \mathbb{R}^{d_{\text{model}} \times d_z}$.
- Distance $d_{ij} = \|z_i - z_j\|_2$.
- Semantic affinity $a_{ij} = \cos(h_i, h_j)$, where $h_i = Px_i$, $P \in \mathbb{R}^{d_{\text{model}} \times d_a}$.

(국문) ** 잠재 좌표 z** 와 ** 의미 유사도 a_{ij}** 를 통해"가까움 (geometry)"과 "비슷함 (semantics)"을 분리해 모델링합니다.

Neighborhoods:

$$\mathcal{N}_k(i) = \text{Top-}k \text{ of } \{\, j \neq i \;:\; a_{ij} \,\} \quad \text{(semantic top-}k\text{)}, \qquad w_{ij}^{\tau} = \exp\left(-\frac{d_{ij}^2}{\tau}\right).$$

---

3. Alignment Bias β^{align} (정렬)

Intuition. Steer token i to **align** with the **average heading** of its semantic neighbors.

Define a **local heading** for i:

$$u_i = \frac{\sum_{l \in \mathcal{N}_k(i)} \tilde{k}_l}{\left\| \sum_{l \in \mathcal{N}_k(i)} \tilde{k}_l \right\|_2}, \qquad \tilde{k}_l = \frac{k_l}{\|k_l\|_2}.$$

Per-pair alignment score:

$$r_{ij}^{\text{align}} = \tilde{k}_j^{\top} u_i \in [-1, 1], \qquad \beta_{ij}^{\text{align}} = \lambda_{\text{align}} \cdot \gamma_{\text{align}}(i) \cdot r_{ij}^{\text{align}}.$$

Here $\gamma_{\text{align}}(i) = \sigma(\alpha_{\text{align}} \cdot \text{Var}_{l \in \mathcal{N}_k(i)}[\tilde{k}_l])$ gates alignment by neighborhood **coherence** (higher variance → weaker alignment). (국문) 이웃 방향이 한쪽으로 **일관**될수록 정렬 항이 강해지도록 게이팅합니다.

---

4. Separation Bias β^{sep} (분리)

Intuition. Repel **crowding** and **redundancy** near i.

Local density around i:

$$\rho_i = \sum_{l \neq i} w_{il}^{\tau_{\text{sep}}}, \qquad \eta_i = \min\Big(1, \frac{\rho_i}{\kappa}\Big).$$

Redundancy kernel:

$$\phi_{ij}^{\text{red}} = w_{ij}^{\tau_{\text{sep}}} \cdot \max\big(0,\, a_{ij} - \delta\big).$$

Separation bias:

$$\beta_{ij}^{\text{sep}} = -\lambda_{\text{sep}} \cdot \eta_i \cdot \phi_{ij}^{\text{red}}.$$

(국문) ** 밀집도 ρ_i** 가 높을수록, 그리고 ** 의미 중복 a_{ij}** 가 임계 δ 이상일수록 $i \leftrightarrow j$ 연결을 **억제**합니다. 이는 장거리에서의 불필요 상호작용과 근거리 과밀을 동시에 줄입니다.

---

5. Cohesion Bias β^{coh} (응집)

Intuition. Pull i toward its **latent centroid** (group coherence).

Local centroid:

$$c_i = \frac{\sum_l w_{il}^{\tau_{\text{coh}}} z_l}{\sum_l w_{il}^{\tau_{\text{coh}}}}.$$

Cohesion score for (i, j):

$$r_{ij}^{\text{coh}} = -\|z_j - c_i\|_2^2, \qquad \beta_{ij}^{\text{coh}} = \lambda_{\text{coh}} \cdot \gamma_{\text{coh}}(i) \cdot \frac{r_{ij}^{\text{coh}}}{\tau_{\text{coh}}},$$

with $\gamma_{\text{coh}}(i) = \sigma\big(\alpha_{\text{coh}} \cdot \text{Var}_l[z_l]\big)$, attenuating when the local manifold is too scattered.

(국문) j 가 i 의 ** 응집 중심 c_i** 에 가까울수록 보너스를 받아 **의미 군집**을 형성합니다.

6. Normalization & Stability (정규화·안정화)

Per-row zero-mean, unit-variance normalization prevents bias drift:

$$\tilde{\beta}^{\star}_{ij} = \frac{\beta^{\star}_{ij} - \mu^{\star}_{i}}{\sigma^{\star}_{i} + \varepsilon}, \quad \mu^{\star}_{i} = \frac{1}{n}\sum_{j}\beta^{\star}_{ij}, \;\; \sigma^{\star}_{i} = \sqrt{\frac{1}{n}\sum_{j}(\beta^{\star}_{ij} - \mu^{\star}_{i})^{2}},$$

for $\star \in \{\text{align}, \text{sep}, \text{coh}\}$. Use a shared **temperature** τ_{score} (or head-specific) before softmax:

$$S_{ij} = \frac{q_i^{\top} k_j}{\sqrt{d}} + \sum_{\star} \omega_{\star} \tilde{\beta}^{\star}_{ij}, \qquad A_{ij} = \text{softmax}_j(S_{ij}/\tau_{\text{score}}).$$

(국문) 각 편향의 **행 (토큰 i) 별 정규화**로 수렴 안정성을 확보하고, $\omega_{\star}$ 는 학습 가능한 스칼라 (또는 헤드별) 가중치입니다.

---

7. Multi-Head Integration (멀티헤드 통합)

Each head h has its own $\{U_h, P_h, \lambda_{\star,h}, \tau_{\star,h}, k_h\}$. **Specialization** emerges: some heads may emphasize Separation (denoising), others Cohesion (grouping), others Alignment (directional flow).

(국문) 헤드별로 다른 스케일·이웃 크기·온도를 두어 **역할 분화**를 유도합니다.

---

8. Training & Regularization (학습·정규화 항, 선택)

Optional auxiliary losses to **shape** the biases:

- **Compactness vs. Spread:**

$$\mathcal{L}_{\text{coh}} = \frac{1}{n}\sum_{i} \mathrm{E}_{j\sim A_i}\big[\|z_j - c_i\|_2^2\big].$$

- **Crowding Penalty:**

$$\mathcal{L}_{\text{sep}} = \frac{1}{n}\sum_{i}\sum_{j} A_{ij}\, \phi^{\text{red}}_{ij}.$$

- **Alignment Smoothness:**

$$\mathcal{L}_{\text{align}} = \frac{1}{n}\sum_i \Big(1 - u_i^\top \bar{u}_{\mathcal{N}(i)}\Big), \quad \bar{u}_{\mathcal{N}(i)} = \frac{1}{|\mathcal{N}_k(i)|}\sum_{l\in\mathcal{N}_k(i)} u_l.$$

Total loss: $\mathcal{L} = \mathcal{L}_{\text{task}} + \alpha_1\mathcal{L}_{\text{coh}} + \alpha_2\mathcal{L}_{\text{sep}} + \alpha_3\mathcal{L}_{\text{align}}$ (all α optional).

(국문) 과도한 규제는 피하고, **warm-up 이후**에 가볍게 거는 것을 권장합니다.

---

9. Computational Complexity (복잡도)

- Base attention: $O(n^2 d)$.
- ASCender overhead per head:
  - Build $\mathcal{N}_k(i)$: Top-k by a_{ij}.
    - From base scores: reuse top-k neighbors (no extra n^2 pass), or
    - Local window w (sequence) $\rightarrow O(nw)$.
  - Bias computations: $O(nkd)$ for headings/centroids; kernels $O(nk)$.

Practical recipe. Use **windowed** + **few global** tokens to cap k, compute z, P with lightweight projections, and fuse bias kernels into the pre-softmax kernel (FlashAttention-compatible).

(국문) 희소/선형/플래시 계열과 **직교**적으로 결합 가능하며, 구현은"pre-softmax addend"로 넣으면 됩니다.

---

10. Pseudocode (의사코드)

Per head h:

```
Inputs: X (n x d_model), WQ, WK, WV, U, P, params {k, tau_sep, tau_coh,
        lambda_align, lambda_sep, lambda_coh, omega_*}

Q = X WQ; K = X WK; V = X WV
Z = X U        # latent coords
H = X P        # semantic proj for affinity

# (1) base scores (optionally add standard pos. biases)
S_base = (Q @ K^T) / sqrt(d)

# (2) neighborhoods & kernels
a = cosine(H, H)                    # (n x n) or windowed
```

```
N_k = topk_indices(a, k)            # per row
d2 = pairwise_sqdist(Z, Z)          # windowed if long seq
w_sep = exp(-d2 / tau_sep); w_coh = exp(-d2 / tau_coh)

# (3) Alignment
k_hat = normalize_rows(K)
u = normalize_rows(sum_over_neighbors(k_hat, N_k))
r_align = row_dot(k_hat, u)      # r_align[i,j] = k_hat[j]·u[i]
beta_align = lambda_align * gate_align(u, N_k) * r_align

# (4) Separation
rho = row_sum(w_sep)                # local density
eta = clip(rho / kappa, 0, 1)
phi_red = w_sep * relu(a - delta)
beta_sep = - lambda_sep * outer(eta, 1s) * phi_red

# (5) Cohesion
c = rowwise_weighted_centroid(Z, w_coh)   # c[i] = Σ w_coh[i,l] Z[l] / Σ w_coh[i,l]
r_coh = - rowwise_sqdist_to(Z, c)           # r_coh[i,j] = -
||Z[j]-c[i]||^2
beta_coh = (lambda_coh / tau_coh) * gate_coh(Z) * r_coh

# (6) Normalize and combine
beta_align = row_norm(beta_align); beta_sep = row_norm(beta_sep); beta_coh = row_norm(be
S = S_base + omega_a*beta_align + omega_s*beta_sep + omega_c*beta_coh
A = softmax(S / tau_score)
Y = A @ V
return Y
```

---

11. Compatibility & Ablations (호환·어블레이션)

- **Compatibility:** ALiBi/relative bias → add as $\beta^{(\text{other})}$; FlashAttention → fuse addends; Long-context memory → ASCender biases computed on the active window + memories.
- **Ablations:**
  (i) remove each bias; (ii) vary k, τ; (iii) head specialization (freeze one bias per head);
  (ii) measure interpretability by decomposing $S = S^{\text{base}} + \sum_{\star} \omega_{\star} \tilde{\beta}^{\star}$ and visualizing per-$\star$ attention maps;
  (iii) efficiency with/without neighborhood reuse.

(국문) 해석 가능성은 **항목별 점수 분해 시각화**로 정량·정성 평가합니다.

---

12. Implementation Notes (구현 메모)

- Start with small $\omega_\star$ and **warm-up**; enable λ_{sep} slightly later to avoid early under-attention.
- Clip $\|u_i\|$ numerically; add ε in norms.
- Cache $\mathcal{N}_k(i)$ from base top-k to avoid extra passes.
- For vision/graph: replace sequence windows with spatial/graph neighborhoods (same formulas).

(국문) 초기 학습 안정화를 위해 정렬/응집을 먼저 활성화하고, 분리는 **후행** 적용을 권장합니다.

Experiments

(KR) 연구 질문

- **RQ1 (정확도):** ASCender 편향이 강력한 기본 Transformer 대비 성능을 향상시키는가?
- **RQ2 (효율성):** 동일 품질 기준에서 불필요 상호작용·FLOPs·시간을 줄이는가?
- **RQ3 (해석 가능성):** 점수 분해맵이 사람이 직관하는 군집·관계와 부합하는가?
- **RQ4 (호환성):** 위치 인코딩·효율 커널과 **직교적**으로 결합 가능한가?

(KR) 벤치마크·지표

- **추론/수학:** GSM8K, MATH (정확도/EM; 기본은 CoT 미사용, 부록에 CoT).
- **언어 이해:** GLUE, SuperGLUE(표준 지표).
- **장문:** LRA / LongBench / SCROLLS(과제별 지표 + 지연/메모리).
- **그룹핑 (선택):** 객체/슬롯 기반 군집의사라벨과 NMI/ARI.
- **효율:** Attention FLOPs, 피크 메모리, 처리속도 (tokens/sec), Flash/희소 결합 전후.

(KR) 비교 대상

1. **기본:** 바닐라 Transformer(+ RoPE 또는 ALiBi).
2. **효율:** Longformer/BigBird(희소), Performer/Linear(커널), FlashAttention(커널 융합).
3. **구조 편향:** 상대 거리 바이어스, (선택) 구문/그래프 편향.

우리 모델:

- **ASCender-Base:** 기본 + 정렬/분리/응집.
- **결합 실험:** ASCender + RoPE/ALiBi, ASCender + 희소/Flash.

(KR) 학습 설정

- **모델 크기:** Small/Medium 에서 탐색 후 Large 확인.
- **최적화:** AdamW, 워밍업, 코사인 감쇠.
- **ASCender 스케줄:** 정렬/응집은 초반부터, **분리**는 워밍업 이후 활성화.
- **이웃:** 국소 창 w+ 글로벌 소수, $k \in \{8, 16, 32\}$, $w \in \{64, 128, 256\}$.
- **온도:** $\tau_{\text{sep}}, \tau_{\text{coh}} \in \{0.5, 1, 2\}$; $\tau_{\text{score}} \in \{0.7, 1.0\}$.
- **게이트/가중치:** $\omega_\star$ 작게 시작 → 학습; 정규화·클리핑 적용.

(KR) 어블레이션·분석

- 편향 항 제거 실험 (단일·조합), $k/w/\tau$ 변화, top-k 재사용 여부.
- 헤드 특화 vs. 공유.
- **해석:** S^{base} 및 $\tilde{\beta}$ 시각화, 군집 지표 (NMI/ARI).
- **통계:** 10 회 반복, 평균 $\pm$ 표준편차, 부트스트랩/랜덤화 검정.

(KR) 재현성

- 시드 다중, 결정적 연산 범위 내 사용.
- 하드웨어/메모리/속도 보고.
- 설정·스크립트 공개, $\omega_\star$ 학습 로그 공개.

PyTorch Reference (concise)

아래는 **헤드 단위 ASCender Attention** 의 간단한 레퍼런스입니다. (실전에서는 Flash/희소 융합, 마스킹, fp16 안정화 등을 추가하세요.)

```
import torch
import torch.nn as nn
import torch.nn.functional as F

def cosine_similarity(x, y, eps=1e-6):
    x = F.normalize(x, dim=-1)
    y = F.normalize(y, dim=-1)
    return x @ y.transpose(-2, -1)

class ASCenderHead(nn.Module):
    def __init__(self, d_model, d_head, k=16, tau_sep=1.0, tau_coh=1.0,
                 lambda_align=1.0, lambda_sep=1.0, lambda_coh=1.0):
        super().__init__()
        self.WQ = nn.Linear(d_model, d_head, bias=False)
        self.WK = nn.Linear(d_model, d_head, bias=False)
        self.WV = nn.Linear(d_model, d_head, bias=False)
        self.U  = nn.Linear(d_model, d_head // 2, bias=False)   # latent Z
        self.P  = nn.Linear(d_model, d_head // 2, bias=False)   # semantic H
```

```
        self.k = k
        self.tau_sep = tau_sep
        self.tau_coh = tau_coh
        self.lambda_align = nn.Parameter(torch.tensor(lambda_align, dtype=torch.float3
        self.lambda_sep   = nn.Parameter(torch.tensor(lambda_sep,   dtype=torch.float3
        self.lambda_coh   = nn.Parameter(torch.tensor(lambda_coh,   dtype=torch.float3
        self.omega_a = nn.Parameter(torch.tensor(0.1, dtype=torch.float32))
        self.omega_s = nn.Parameter(torch.tensor(0.1, dtype=torch.float32))
        self.omega_c = nn.Parameter(torch.tensor(0.1, dtype=torch.float32))
        self.score_temp = nn.Parameter(torch.tensor(1.0, dtype=torch.float32))
        self.delta = nn.Parameter(torch.tensor(0.2, dtype=torch.float32))  # redundanc
        self.kappa = nn.Parameter(torch.tensor(32.0, dtype=torch.float32)) # density s

    def row_norm(self, B, eps=1e-6):
        mu = B.mean(dim=-1, keepdim=True)
        sd = B.std(dim=-1, keepdim=True) + eps
        return (B - mu) / sd

    def forward(self, X, attn_mask=None):
        # X: [B, N, d_model]
        B, N, _ = X.shape
        Q = self.WQ(X)   # [B,N,d]
        K = self.WK(X)
        V = self.WV(X)
        Z = self.U(X)    # latent coords
        H = self.P(X)    # semantic proj

        # base scores
        d = Q.size(-1)
        S_base = (Q @ K.transpose(-2, -1)) / (d ** 0.5)  # [B,N,N]

        # semantic neighborhoods (top-k by cosine on H)
        Asem = cosine_similarity(H, H)                    # [B,N,N]
        topk_val, topk_idx = torch.topk(Asem, k=min(self.k, N), dim=-1)  # [B,N,k]

        # unit K for alignment
        K_hat = F.normalize(K, dim=-1)                    # [B,N,d]
        # compute local heading u_i = normalize(sum_{l in N_k(i)} k_hat_l)
        u = torch.zeros_like(K_hat)
        gather_K = K_hat.unsqueeze(2).expand(B, N, N, d).gather(
            2, topk_idx.unsqueeze(-1).expand(B, N, self.k, d)
        )                                                 # [B,N,k,d]
        u = F.normalize(gather_K.sum(dim=2), dim=-1)    # [B,N,d]
        # r_align[i,j] = k_hat[j] · u[i]
        r_align = (K_hat @ u.transpose(-2, -1))         # [B,N,N]
        beta_align = self.lambda_align * r_align        # gate by variance can be adde
```

```
        # distances for sep/coh
        # pairwise ||Z_i - Z_j||^2 = ||Z_i||^2 + ||Z_j||^2 - 2 Z_i·Z_j
        Zi2 = (Z**2).sum(dim=-1, keepdim=True)          # [B,N,1]
        Zj2 = (Z**2).sum(dim=-1).unsqueeze(1)           # [B,1,N]
        d2 = Zi2 + Zj2 - 2 * (Z @ Z.transpose(-2, -1))  # [B,N,N]

        w_sep = torch.exp(-d2 / (self.tau_sep + 1e-6))
        w_coh = torch.exp(-d2 / (self.tau_coh + 1e-6))

        # local density rho_i = sum_j w_sep[i,j]
        rho = w_sep.sum(dim=-1, keepdim=True)           # [B,N,1]
        eta = torch.clamp(rho / (self.kappa + 1e-6), 0.0, 1.0)  # [B,N,1]

        # redundancy kernel phi_red = w_sep * relu(a_ij - delta)
        phi_red = w_sep * F.relu(Asem - self.delta)
        beta_sep = - self.lambda_sep * eta * phi_red    # broadcast over j

        # cohesion: c_i = Σ w_coh[i,l] Z_l / Σ w_coh[i,l]
        denom = w_coh.sum(dim=-1, keepdim=True) + 1e-6
        c = (w_coh @ Z) / denom                         # [B,N,dz]
        # r_coh[i,j] = -||Z_j - c_i||^2
        r_coh = -((Z.unsqueeze(1) - c.unsqueeze(2))**2).sum(dim=-1)  # [B,N,N]
        beta_coh = self.lambda_coh * r_coh / (self.tau_coh + 1e-6)

        # normalize each bias row-wise
        beta_align = self.row_norm(beta_align)
        beta_sep   = self.row_norm(beta_sep)
        beta_coh   = self.row_norm(beta_coh)

        S = S_base + self.omega_a*beta_align + self.omega_s*beta_sep + self.omega_c*be

        if attn_mask is not None:
            S = S.masked_fill(attn_mask == 0, float('-inf'))

        A = torch.softmax(S / (self.score_temp + 1e-6), dim=-1)
        Y = A @ V
        return Y, A, dict(S_base=S_base, b_align=beta_align, b_sep=beta_sep, b_coh=bet
```

메모

- 실전에서는 **윈도우 마스크** + **소수 글로벌 토큰**을 적용해 N^2 를 제한하세요.
- FlashAttention 사용 시 위의 `S` 합산을"pre-softmax addend"로 합치면 됩니다.
- 시각화를 위해 `dict(...)` 로 분해 항을 리턴해 두면 편합니다.

1) Results Tables (EN/KR)

(KR) 효율/지연/메모리

표 2: 효율 지표 (동일 품질 목표선에서 측정).

모델	Attention FLOPs	Peak Memory (GB)	Tokens/sec	Latency (ms)
Transformer (base)	XXXX	X.XX	XXXX	XXX
ASCender (ours)	XXXX	X.XX	XXXX	XXX
ASC + Sparse/Flash	XXXX	X.XX	XXXX	XXX

주: 동일 파라미터 수/배치/시퀀스 길이/정확도 범위 내 맞춘 공정 비교.

(KR) 해석 가능성/군집 지표

표 4: 주의 맵 분해 기반 군집 품질 (의사 라벨 또는 보조 태스크 라벨 대비).

모델	NMI ↑	ARI ↑	Calinski-Harabasz ↑	Silhouette ↑
Transformer (base)	X.XX	X.XX	XXXX	X.XX
ASCender (ours)	X.XX	X.XX	XXXX	X.XX
ASC (ablations)	X.XX	X.XX	XXXX	X.XX

---

2) Figure Guide (Layouts & Captions)

Fig. 1 —Decomposed attention maps (EN)

What: Show S^{base}, $\tilde{\beta}^{\text{align}}$, $\tilde{\beta}^{\text{sep}}$, $\tilde{\beta}^{\text{coh}}$, and final S. **Caption template:** Figure 1: Score decomposition for a representative example. ASCender's Alignment emphasizes semantically similar neighbors, Separation suppresses redundant/local crowding, and Cohesion consolidates groups around latent centroids. The combined score S yields sharper, interpretable attention patterns.

Fig. 2 —Pareto: quality vs. cost (KR)

내용: 정확도/EM 등 품질 축 vs. FLOPs/지연/메모리 축. ASCender 가 **Pareto frontier** 를 확장하는지 시각화. **캡션:** 그림 2: 품질–비용 파레토 전선. ASCender 는 동일 품질에서 비용을 감소시키거나, 동일 비용에서 품질을 향상하여 전선을 외측으로 확장한다.

Fig. 3 —Ablation (EN)

What: Remove A/S/C individually and jointly; plot accuracy and efficiency deltas. **Caption:** Figure 3: Ablation of ASCender biases. Separation primarily contributes to efficiency (fewer redundant interactions), while Alignment and Cohesion drive accuracy through semantically coherent grouping.

Fig. 4 —Head specialization heatmap (KR)

내용: 헤드별 $\omega_\star$ 또는 편향 기여도 분포 히트맵, 시드 평균. **캡션:** 그림 4: 헤드 전문화. 일부 헤드는 분리 (잡음 억제) 에, 일부는 응집 (그룹 형성), 일부는 정렬 (방향성) 에 특화되는 경향을 보인다.

Fig. 5 —Long-context scaling (optional, EN)

What: Accuracy vs. sequence length (e.g., 1k, 4k, 16k). **Caption:** Figure 5: Length scaling. ASCender maintains quality under increasing sequence lengths by suppressing irrelevant long-range links and reinforcing salient clusters.

---

3) Statistical Reporting Checklist (KR 중심)

- **반복 실험:** 시드 ≥ 5, `mean ± std` 보고.
- **유의성 검정:** 페어드 부트스트랩 (테스트셋 재표본, 10k 이상) 또는 Approx. Randomization.
- **효과 크기:** Cliff's delta 또는 Cohen's d 병기 권장.
- **동일 조건:** 파라미터 수, 토큰화, 학습 스텝/예산, 정지 조건 (early stop) 명시.
- **자원 보고:** GPU/CPU 사양, 메모리 한계, 벽시계 시간.
- **학습 곡선:** 검증 점수 vs. 스텝 커브 (평활화 X), ASCender 편향 가중치 $\omega_\star$ 의 학습 추이도 첨부.
- **해석 가능성:** 분해 맵의 정량 지표 (NMI/ARI) + 정성 사례 (본문/부록) 병행.

---

4) Conclusion (EN/KR)

(KR) 결론 (간결판)

본 연구는 `Boids` 의 정렬·분리·응집 원리를 `Attention` 점수에 직접 주입하는 `ASCender` 를 제안하였다.
추론·언어·장문 벤치마크에서 `ASCender` 는 품질을 향상시키면서 효율–품질 균형을 개선하며, 점수 분해

사람이 이해 가능한 군집 구조를 드러낸다. 분석 결과, 분리는 중복 상호작용을 억제해 효율에 기여하고
정렬·응집은 의미적 군집을 형성해 성능을 끌어올리며, 헤드별 역할 분화가 관찰되었다. 또한 RoPE/ALi
희소/Flash 계열과 직교적으로 결합 가능해 적용 범위가 넓다.

한계로는 이웃 크기·온도 등 하이퍼파라미터 튜닝 부담과 초반 학습에서 분리 항 과대 시 불안정성이 있
향후 연구로는 태스크 적응형 바이어스 스케줄링, 구문/그래프 등 외부 구조 단서와의 결합, 멀티모달·
검색 결합 환경으로의 확장을 제안한다.

---

5) (Optional) Appendix Stubs (붙이기 좋은 부록 틀)

- **A. Implementation details:** optimizer, LR schedule, mixed precision, gradient clipping, masks, caching of $\mathcal{N}_k$.
- **B. Additional ablations:** k, w, τ sweeps, gates on/off, head freezing, memory tables.
- **C. Visualization protocol:** how examples are chosen, seeds, colorbars, resolution, failure cases.
- **D. Statistical methods:** exact bootstrap/randomization procedures and code pointers.
- **E. Ethics/Limitations:** deployment considerations, bias risks, compute footprint.