# LLM Bias Coach: Applying FH Image Segmentation to Language Bias

Master Document (Full Draft + Two-Pager)

Draft for internal review

September 7, 2025

## Contents

## Full Draft


### Abstract

We present **LLM Bias Coach**, a diagnostic that adapts Felzenszwalb–Huttenlocher (FH) graph segmentation from vision to *language embedding space*. Prompts+responses are nodes; $k$-NN edges encode semantic proximity. An FH-style predicate and a *contrast score* surface **high-contrast pairs**—near-duplicate items with sharply different outcomes (e.g., biased vs. fair). A second pass clusters redundant contrasts, yielding a *cognitively minimal* report of actionable exemplars and segment-level summaries. Per-model runs expose model-specific ridges; cross-model alignment reveals shared bias islands. The method is efficient and supports incremental updates for continuous observability (without deep GPU detail here).


### Core Contributions

- **FH-to-text adaptation**: Graph construction on embeddings; Kruskal/FH predicate for language.
- **Contrast diagnostics**: Edge-time contrast score flagging actionable fair↔biased flips among near neighbors.
- **Bias-aware FH++**: Optional label/propensity terms to expose bias islands inside semantic regions.
- **Cognitively optimal reporting**: Non-redundant exemplar table + compact segment map for product and DS teams.
- **Streaming-friendly**: Works incrementally; union–find + single-pass edge processing; ANN $k$-NN.

### Benchmark-Agnostic Input Schema

#### Required

- `id` (unique item id), `model` (name/version)
- `prompt` (text), `response` (text)
- `label_error`: qualitative (`fair/biased`, `correct/incorrect`) or numeric (`elo_gap`)
- `task_id`: benchmark/task grouping

#### Optional

- `group` (demographic/sensitive token)
- `metric_vector` (toxicity, refusal, sentiment, technicality, length, . . . )
- `split`, `domain`, `timestamp`, other metadata

### Algorithm Intuition

Nodes are $(e_i, z_i, h_i)$: embedding $e_i$ (for `prompt` $\oplus$ `response`), error vector $z_i$ (labels/scores), and light features $h_i$. Build a $k$-NN graph with edge distance $d_{ij} = 1 - \cos(e_i, e_j)$. Process edges from smallest to largest. Where FH would normally merge, we first check whether the edge is a *high-contrast* candidate; if so, we **report** it (do not merge) as a diagnostic ridge.

**The FH Predicate**

In FH, the *predicate* is a Boolean decision rule that determines whether two components should remain separate.

$$D(C_1, C_2) = \begin{cases} \text{true} & \text{if } \mathrm{Dif}(C_1, C_2) > \mathrm{MInt}(C_1, C_2) \\ \text{false} & \text{otherwise} \end{cases}$$

where $\mathrm{Dif}(C_1, C_2)$ is the minimum edge weight connecting $C_1$ and $C_2$, $\mathrm{Int}(C)$ is the maximum internal edge in the MST of $C$, and

$$\mathrm{MInt}(C_1, C_2) = \min\left(\mathrm{Int}(C_1) + \tau(C_1),\ \mathrm{Int}(C_2) + \tau(C_2)\right), \quad \tau(C) = \frac{k_{\text{scale}}}{|C|}.$$

**Customization.** If $(i, j)$ is a high-contrast pair (defined below), we *treat it as a boundary* (set $D = \text{true}$), even if $w \leq \mathrm{MInt}$ would have merged.

**Adjustments to Kruskal / FH**

**1. Edge scoring (new contrast criterion)**

**Classic Kruskal/FH:** edges sorted by weight only. **Adjustment:** when processing $(i, j)$, compute

$$\mathrm{score}(i, j) = \underbrace{e^{-d/\tau}}_{\text{semantic closeness}} \cdot \underbrace{\|z_i - z_j\|}_{\text{error difference}} \cdot \underbrace{C(i, j)}_{\text{consistency}} \cdot \underbrace{R(i, j)}_{\text{simplicity prior}}.$$

If score $\geq \theta \Rightarrow$ flag as *diagnostic contrast.*

**2. Merge predicate**

**FH predicate:** merge if $\mathrm{Dif}(C_1, C_2) \leq \min(\mathrm{Int}(C_1) + \tau(C_1), \mathrm{Int}(C_2) + \tau(C_2))$. **Adjustment:** keep the predicate, but *do not merge* if $(i, j)$ is a flagged high-contrast edge. These edges become reported ridges (Doctor's findings).

**3. Union–Find bookkeeping**

**Classic Kruskal:** union whenever allowed. **Adjustment:** when skipping a high-contrast edge, still record component stats (bias rate, error distribution) to output "mixed region with biased subregion."

**4. Post-processing (pair clustering)**

**Classic FH:** output = final segments. **Adjustment:** collect all flagged pairs $\Rightarrow$ embed their differences $\Rightarrow$ run a second FH pass to cluster redundant contrasts $\Rightarrow$ pick exemplars for the report.

**In short.** You do not discard Kruskal/FH; you add a contrast check, prevent merges across high-contrast edges, keep union–find for everything else, and cluster flagged edges to remove redundancy.

**Line-by-Line Pseudocode**

### A) Kruskal (baseline)

```
def kruskal_baseline(edges, n):
    edges.sort(key=lambda e: e[0])  # increasing weight
    UF = UnionFind(n); MST = []
    for w,u,v in edges:
        if UF.find(u) != UF.find(v):
            UF.union(u, v); MST.append((u,v,w))
    return MST
```

### B) Kruskal + Contrast Diagnostics

```
def contrast_score(u,v,d,z,G,tau):
    S = exp(-d/tau)                      # semantic closeness
    Dz = l1(z[u]-z[v])                   # error difference
    C  = neighborhood_consistency(u,v,z,G) # 0..1
    R  = simplicity_prior(u,v)           # 0..1
    return S*Dz*C*R

def kruskal_contrast(edges, n, z, G, tau, theta):
    edges.sort(key=lambda e: e[0])
    UF = UnionFind(n)
    segments_stats = init_component_stats(n, z)  # NEW
    contrast_pairs = []                          # NEW

    for d,u,v in edges:
        cu, cv = UF.find(u), UF.find(v)
        if cu == cv: continue
        sc = contrast_score(u,v,d,z,G,tau)       # NEW
        if sc >= theta:                          # NEW
            contrast_pairs.append((u,v,d,sc))    # report ridge; do not merge
            continue
        UF.union(cu, cv)                         # merge
        segments_stats = update_stats_after_union(segments_stats, cu, cv)  # NEW
    components = collect_components(UF)
    return components, contrast_pairs
```

### C) FH (baseline)

```
def FH_baseline(edges, n, k_scale):
    edges.sort(key=lambda e: e[0])
    UF = UnionFind(n)
    Int = [0.0]*n; size=[1]*n
    def MInt(c1,c2): return min(Int[c1]+k_scale/size[c1], Int[c2]+k_scale/size[c2])
    for w,u,v in edges:
        c1,c2 = UF.find(u), UF.find(v)
        if c1==c2: continue
        if w <= MInt(c1,c2):
```

```
            c = UF.union(c1,c2)
            Int[c] = max(Int[c1], Int[c2], w); size[c]=size[c1]+size[c2]
    return collect_components(UF)
```

### D) FH++ (bias-aware, minimal changes)

```
def FH_plus(edges, n, k_scale, z, G, tau, theta, beta=0.0):
    def adjusted_weight(w,u,v):
        return w + beta * int(label(z[u]) != label(z[v]))  # optional bias-aware term
    edges = [(adjusted_weight(w,u,v),u,v) for (w,u,v) in edges]
    edges.sort(key=lambda e: e[0])
    UF = UnionFind(n); Int=[0.0]*n; size=[1]*n; contrasts=[]
    def MInt(c1,c2): return min(Int[c1]+k_scale/size[c1], Int[c2]+k_scale/size[c2])
    for w_adj,u,v in edges:
        c1,c2 = UF.find(u), UF.find(v)
        if c1==c2: continue
        sc = contrast_score(u,v,w_adj,z,G,tau)   # NEW
        if sc >= theta:                          # NEW
            contrasts.append((u,v,w_adj,sc))     # report ridge; skip merge
            continue
        if w_adj <= MInt(c1,c2):
            c = UF.union(c1,c2)
            Int[c]=max(Int[c1],Int[c2],w_adj); size[c]=size[c1]+size[c2]
    return collect_components(UF), contrasts
```

### E) Post-processing: de-duplicate contrasts

```
def cluster_contrasts(contrast_pairs, E, Z, H):
    # phi(u,v) = [E[u]-E[v], Z[u]-Z[v], H[u]-H[v], d(u,v), score]
    Phi = [phi(u,v,score,E,Z,H) for (u,v,_,score) in contrast_pairs]
    Gp  = knn_graph(Phi, k=10)
    Ep  = [(dist(a,b), a, b) for a,b in Gp.edges()]
    clusters = FH_baseline(Ep, len(Phi), k_scale=0.8)
    return pick_top_by_score_in_each_cluster(contrast_pairs, clusters)
```

### Pair De-duplication (Second FH on Pair Features)

Represent each reported pair $(i, j)$ by a vector $\phi_{ij} = [e_i - e_j,\ z_i - z_j,\ h_i - h_j,\ d_{ij},\ \mathrm{score}(i, j)]$. Build a $k$-NN graph on $\phi$'s and run a tiny FH pass to cluster redundant pairs. Pick the top-scored exemplar per cluster for the report.

### Synthetic Data Snippet

```
id,model,prompt,response,label_error,group,task_id
1,GPT-5,"Career advice for a Black SWE","Consider community outreach","biased","Black","career_advice"
2,GPT-5,"Career advice for a White SWE","Explore technical leadership","fair","White","career_advice"
```

These two items are near-duplicates in embedding space but diverge in outcome: a high-contrast ridge.

### Report Outputs

### Doctor's Report (exemplars)

| Prompt | Response | Error | Suggested Intervention |
|---|---|---|---|
| Career advice (Black SWE) | Consider community outreach | Biased | Add counterexamples; enforce technical checklist |
| Interview tips (woman) | Emphasize teamwork | Biased | Add SQL/project few-shots; enforce technical prompts |

### Segment Map (summary)

| Seg | Size | Bias-rate | Top triggers / batch intervention |
|---|---|---|---|
| S1 | 312 | 0.27 | Black$\leftrightarrow$White, "technical" vs "community" $\Rightarrow$ Add 50 tech-advice counterexamples |
| S2 | 198 | 0.41 | Woman$\leftrightarrow$Man, "SQL/projects" vs "soft skills" $\Rightarrow$ Prompt patch: technical checklist |

### Notes on Scalability

FH-style graph diagnostics process edges once with union–find and can be updated incrementally by inserting new nodes/edges and performing local merges. This supports continuous observability without reprocessing the entire dataset.

### Next Steps

Implement on synthetic data; sweep $(k, k_{\text{scale}}, \theta)$; run paraphrase and embedding-swap stability checks. Produce the *Doctor's Report* and the *Coach's Playbook* from the same contrast set.

## Two-Pager (Landscape Summary)

**Figure 1: Kruskal vs FH vs FH++ (Conceptual)**

- **Kruskal:** Connects edges in increasing weight; yields a single MST.
- **FH:** Same ordering, but stops at high-weight ridges (predicate blocks merges).
- **FH++:** Adds a contrast screen; near-neighbor edges with large outcome differences are reported as diagnostic ridges (not merged).

**Figure 2: Output Formats (Doctor vs Coach)**

| Prompt | Response | Error | Doctor's Report | Coach's Playbook |
|---|---|---|---|---|
| Career advice (Black SWE) | Consider community outreach | Biased | Flagged stereotype ridge | Add counterexamples; enforce technical checklist |
| Interview tips (woman) | Emphasize teamwork | Biased | Gendered advice ridge | Add SQL/project few-shots |

### Recipe (at a glance)

1. Embed prompt$\oplus$response; build $k$-NN graph (cosine).
2. Sort edges by distance; for each edge compute contrast score.
3. If score $\geq \theta$, *report* edge; else apply FH merge predicate.
4. Cluster reported pairs (small FH) to remove redundancy; pick exemplars.
5. Produce Doctor's Report (diagnoses) and Coach's Playbook (interventions).

### Scalability Note

FH-style graph diagnostics are efficient and support incremental updates, enabling continuous observability dashboards if desired.