– Independent Work Report Fall 2025 –

Reciprocal Compatibility Scoring and Stable Matching: A Transparent Questionnaire-Based Pipeline for Small Campus Cohorts

Ammaar Alam
Adviser: Professor Marcel Dall'Agnol

Abstract

This independent work implements an end-to-end pipeline for reciprocal partner recommendation and cohort-based matching, tailored to small campus settings. Participants complete a structured questionnaire in which they report (i) their own response, (ii) which range or set of responses they would accept from a partner, and (iii) an importance label per item. The system converts these triad responses into directional satisfaction estimates, aggregates them into a symmetric reciprocity-aware compatibility score, and then produces preference lists and matchings via stable matching solvers.

Grounded in the accompanying code repository, we implement and document several compatibility models: (1) a deterministic baseline that aggregates weighted hard acceptability with domain multipliers, a user-specified priority boost, an overlap filter, and a finite-overlap penalty; (2) a probabilistic acceptability model (PAM) that uses response-type-specific acceptability kernels and uncertainty-aware lower confidence bounds (LCBs); (3) interpretable domain-wise distance-kernel models, including a configurable scorer, a soft-gated learner trained with a pairwise ranking objective, and an evolutionary configuration search procedure; and (4) a merged inverse document frequency (IDF) + LCB pipeline that constructs tie-aware preference tiers for downstream matching.

Because Institutional Review Board (IRB) approval is pending, this draft reports no human-subject empirical results. Instead, it specifies an evaluation framework for comparing scoring models and downstream matching solvers using offline ranking, separation, and stability diagnostics once IRB-approved data collection begins. Source code and non-sensitive artifacts are available at `https://github.com/Ammaar-Alam/matching-algorithm-repo`.

1. Introduction

Reciprocal recommendation differs from standard recommendation because a suggested match is only useful if *both* people find each other acceptable. In a campus dating context, this reciprocal constraint is coupled with system-level questions: how to (i) rank potential partners for a user in real time, and (ii) periodically form stable matchings from a cohort to facilitate introductions.

The gap. Commercial dating apps like Tinder and Hinge use opaque machine learning models to optimize for user engagement based on behavioral signals, including millions of swipes, messages, and time spent on profiles. These methods are ill-suited to smaller campuses, since there are fewer observations, a higher level of privacy is expected, and a greater social cost is associated with any mistakes made through frequent interactions. Even OkCupid, whose personality-profile questionnaire inspired the concept of user-facing compatibility scores (and ultimately our triad questionnaire), operates at a scale and data-collection model incompatible with sensitive campus settings and has few published reliability or validity outcomes.

Our approach. Our work develops a transparent, uncertainty-aware pipeline that operates primarily from structured questionnaire responses without requiring behavioral logs. The key innovation is a *triad* response format – inspired by OkCupid's question design – where each participant provides (i) their own answer, (ii) which partner answers they would accept, and (iii) an importance weight. This triad representation supports both hard-constraint and probabilistic interpretations, enables domain-level explainability, and naturally interfaces with stable matching algorithms that require preference lists.

The focus of this paper remains on algorithmic compatibility scoring and how those scores interface with matching solvers (preference lists with ties/incompleteness, stable matching, and max-weight baselines), rather than the platforms in which these algorithms could be integrated.

1.1. Design goals and constraints

The project is shaped by four constraints:

Reciprocity. A match is successful only if both sides would accept each other. This motivates (i) directional satisfaction scores (how much u likes v) and (ii) symmetric aggregation rules that penalize one-sided compatibility.

Transparency and explainability. Participants should understand what inputs drive recommendations. This motivates explicit domain structure (values, lifestyle, etc.) and scoring functions that can be decomposed into interpretable terms rather than end-to-end black boxes.

Small data. In early deployments, there may be few participants and no behavioral logs. This motivates conservative scoring (e.g., confidence bounds) and learning procedures that explicitly control overfitting.

Social and ethical sensitivity. Dating recommendations can cause embarrassment, reinforce bias, or create safety issues. This motivates careful data handling, consent considerations, and avoiding claims beyond the evidence.

1.2. IRB status and scope of this draft

Our work is currently constrained by ethics and logistics. IRB approval is pending, which blocks recruitment and the reporting of human-subject empirical results in this draft. Accordingly, the focus here is to (i) formalize the end-to-end scoring and matching pipeline and (ii) specify an evaluation framework that can be executed verbatim once IRB-approved data collection begins.

During development, we validate the end-to-end code path (questionnaire ingestion $\rightarrow$ pairwise scores $\rightarrow$ metrics and matchings) using synthetic inputs and unit tests. These tests also help surface failure modes that inform future data collection and model design.

What this report does *not* claim. This report does not claim real-world effectiveness, statistical significance, or generalization. It documents an implementation and a planned evaluation protocol.

1.3. Comparison axes and evaluation metrics

The repository defines an end-to-end pipeline with two conceptually separable components: (i) a *scoring model* that maps two questionnaire responses to a reciprocity-aware compatibility score (or a tiered preference list), and (ii) a *matching solver* that consumes scores/lists to produce cohort-level outcomes (a stable matching, a maximum-weight matching, etc.).

Our comparisons therefore vary two axes:

- **Scoring model family.** Baseline FinalMatch (hard acceptability), PAM+LCB (probabilistic acceptability with uncertainty), distance-kernel models (configurable / soft-gated / evolutionary), and a merged *inverse document frequency* (IDF) + LCB tiering pipeline.
- **Matching solver.** Deferred acceptance (DA), stable roommates, maximum-weight matching (blossom), and a MILP "best stable" solver.

All variants are evaluated with a common family of offline metrics (Section 6.2): rank-based partner-retrieval metrics (Hit@K, MRR, Mutual@K, rank statistics), true-vs-random separation metrics, *area under curve* (AUC) and lift, and, when a matching is produced, matching-level diagnostics (match rate, total weight, and blocking-pair counts under the induced preferences).

Parameters: fixed, tunable, learned, compared. To make comparisons unambiguous, we group knobs into four categories:

- **Fixed throughout:** questionnaire specification (`data/questions.json`), domain labels $d(q)$, response parsing rules, missingness handling, and the cohort being scored.
- **Default but tunable (held fixed unless ablated):** baseline domain multipliers and priority boost; importance mappings; PAM kernel parameters and LCB confidence level; merged-pipeline tie and soft-cap thresholds.
- **Learned from data:** parameters of the soft-gated model, optional learned domain weights, and the evolutionary configuration (mode/weights/kernel parameters per domain).
- **Explicitly compared:** (i) scoring-model choice, (ii) baseline hyperparameter variants (e.g., overlap penalty and aggregation rule), and (iii) downstream matching-solver choice.

1.4. Contributions (implementation-focused)

Within the above constraints, this paper delivers:

1. A formalization of the questionnaire-to-score pipeline implemented in the repository, including

Table 1: What is compared in TigerMatch. Scoring models produce pairwise scores (or tiered preferences) from questionnaires; matching solvers consume these to produce cohort-level matchings.

Component	Output	Example knobs	Learned?
Baseline FinalMatch	score matrix	c, $n_{\min}$, arithmetic vs geometric	no
PAM+LCB	score + uncertainty	kernel params, confidence level, importance map	optional
Distance-kernel (config)	score matrix	per-domain mode, (μ, σ) grids	no
Soft-gated	score matrix	regularization, optimizer settings	yes
Evolutionary search	score matrix	objective weights, search budget	yes (via search)
Merged IDF+LCB	tiered preferences	IDF prior, softcap ρ, tie threshold τ	no
Matching solvers	matching	DA / roommates / blossom / best-stable MILP	no

notation and code-grounded formulas for directional and reciprocal scoring.

2. A deterministic baseline compatibility function (`algorithm/algorithms/core.py::final_match`) with multiple hyperparameter variants (`algorithm/algorithms/variants.py`).
3. A probabilistic acceptability model (PAM) with response-type kernels, importance compression, and LCB-based uncertainty handling (Section 4.3).
4. Domain-wise distance-kernel scoring and learning procedures (soft-gated training and evolutionary search) grounded in `algorithm/features`, `algorithm/scoring`, and `algorithm/ml`.
5. A merged IDF+LCB pipeline that produces tie-aware preference tiers and connects to stable matching solvers (`algorithm/matching/*`).
6. An evaluation harness and reporting framework for comparing scoring models and matching solvers using offline ranking, separation, and stability diagnostics (Section 6).

1.5. Organization

Section 2 reviews matching markets, reciprocal recommendation, and uncertainty-aware scoring. Section 3 describes the data and ethical constraints. Section 4 presents scoring and matching algorithms. Section 5 summarizes implementation details. Section 6 specifies the evaluation framework and metrics. Section 7 discusses limitations and ethical considerations.

2. Background and Related Work

TIGERMATCH sits in the middle of matching theory and reciprocal recommendation. This section reviews prior work to situate design choices and to clarify where the project departs from established settings.

2.1. Stable matching and stable roommates

Matching theory studies allocation problems where participants have preferences over potential partners. In the classical bipartite setting ("stable marriage" / college admissions), the deferred acceptance algorithm of Gale and Shapley [1962] produces a stable matching and has become foundational for market design. In many applications, stability is interpreted as the absence of a *blocking pair*: two participants who would both prefer each other over their assigned outcomes.

Campus dating does not naturally partition participants into two disjoint sides, motivating the *stable roommates problem* in a single population. Irving [1985] gives a polynomial-time algorithm for stable roommates under strict preferences. In practice, preference lists may include ties and incompleteness (participants find some partners unacceptable), connecting to the *stable marriage with ties and incomplete lists* (SMTI) literature [Iwama et al., 1999].

2.2. Preferences derived from scores: ties and incompleteness

In classical matching theory, preferences are typically taken as primitive inputs: each agent submits a strict ranking. In TIGERMATCH, preferences are *derived* from a score function $S(u, v)$ computed from questionnaire responses. This creates two practical issues.

Ties. If scores are coarse (for example, when many candidates satisfy the same set of acceptability constraints), then ties are unavoidable. Ties matter because stability definitions assume strict preferences. There are multiple ways to handle ties: (i) break ties arbitrarily (which may change the outcome), (ii) treat ties as indifference and use a stability notion for weak preferences, or (iii) incorporate ties directly into an optimization problem. TIGERMATCH primarily uses deterministic tie-breaking for algorithms that require strict lists, but it also constructs explicit tie classes in the merged pipeline to make the presence of near-equivalent candidates visible.

Incomplete lists. A participant may deem many candidates unacceptable. In stable matching, incomplete lists are standard: an agent can rank only acceptable partners and remain unmatched otherwise. In a campus matching system, incompleteness is not just a modeling convenience; it's ethically important. Forcing a participant to be matched to someone they find unacceptable can cause harm. TIGERMATCH therefore treats acceptability as a first-class concept: candidates that violate mandatory constraints can be assigned very low scores or removed from the list entirely.

2.3. Polyhedral structure and optimization over stable outcomes

Beyond existence and construction, stable matchings admit a polyhedral characterization. Vande Vate [1989] and subsequent work (including Rothblum [1992]) characterize stable matchings via linear constraints. This supports optimization *over* stable matchings when multiple stable matchings exist (e.g., maximizing total weight subject to stability), and it motivates mixed-integer or linear programming approaches.

The repository includes an LP/MILP-style solver for selecting a high-weight stable matching after generating preference lists (`algorithm/algorithms/best_stable_lp.py`), aligning with this perspective.

2.4. Incentives and manipulation

Stable matching mechanisms raise incentive issues: participants may strategically misreport preferences to obtain better outcomes. In the bipartite DA setting, strategy-proofness holds for one side but not necessarily the other. Classical results such as Dubins and Freedman [1981] and Roth [1982] motivate caution when deploying mechanisms that elicit preferences directly. While this draft does not empirically study manipulation (IRB-approved data collection is pending), these considerations are important for future deployment.

2.5. Reciprocal recommender systems

Dating recommendation is an instance of *reciprocal* recommendation, where recommendation quality depends on mutual interest. RECON [Pizzato et al., 2010] is a notable early system that explicitly models reciprocity in online dating. Xia et al. [2016] discuss design principles for reciprocal recommenders, emphasizing the differences from unilateral recommendation and the need for balancing both sides' satisfaction.

Questionnaire-based approaches and OkCupid. OkCupid popularized a questionnaire-based approach where users answer questions, specify acceptable partner answers, and indicate importance – a *triad* format that directly inspired TIGERMATCH's data representation.[1] Their public writings describe computing compatibility as weighted agreement over questions where both users care about the answer. However, OkCupid operates at massive scale with proprietary algorithms and extensive behavioral data; their approach is not directly transferable to a small, privacy-sensitive campus setting. TIGERMATCH adapts the triad concept but adds uncertainty-aware scoring (LCBs), domain-level structure, and explicit interfaces to stable matching solvers.

The questionnaire items themselves draw on established psychometric instruments. Personality items are structured after the Ten-Item Personality Inventory [TIPI; Gosling et al., 2003], a brief Big Five measure validated for contexts where time is limited. Values items draw on the Schwartz theory of basic values and the Portrait Values Questionnaire [PVQ; Schwartz, 2012, Schwartz et al., 2001], which provides a well-validated framework for assessing personal value priorities. Using established instruments as templates improves content validity and connects the questionnaire to a broader empirical literature on personality and values in relationships.

This paper's scoring functions explicitly compute directional satisfaction and then aggregate to a symmetric reciprocal score, mirroring this basic requirement.

[1] See C. Rudder, *Dataclysm*, Crown, 2014, for public descriptions of OkCupid's matching methodology.

2.6. Similarity versus complementarity

Relationship research and common intuition suggest competing hypotheses about what makes a good match. Similarity models reward small distances in traits and values; complementarity models reward moderate differences. In recommender-system terms, these correspond to different kernels over feature distance.

Empirical evidence generally favors similarity, at least for attitudes and values. Montoya et al. [2008] meta-analyzed actual and perceived similarity, finding that actual similarity is positively associated with attraction. Luo and Klohnen [2005] found that newlywed couples showed substantial attitude similarity, and Gaunt [2006] linked value similarity to marital satisfaction. However, the picture is more nuanced for personality: Weidmann et al. [2023] found that personality similarity effects on satisfaction are weak to negligible, suggesting that complementarity or irrelevance may be more appropriate for some personality facets.

Rather than choosing a single hypothesis globally, TIGERMATCH uses domain-specific kernels that can be configured as similarity, complementarity, or a learned mixture. This reflects a pragmatic belief: different domains may behave differently. For example, similarity may be more important for values, while complementarity could matter for social preferences.

2.7. Uncertainty-aware decision rules

When datasets are small, uncertainty matters. Confidence bounds are a classic tool for conservative decision-making: they penalize estimates with high variance. TIGERMATCH uses lower confidence bounds (LCBs) to downweight pairs where compatibility is driven by a small number of uncertain items.

Uncertainty-awareness is especially important for transparency. A system that admits “we are not confident about this match” can present results more honestly and can avoid over-emphasizing weak signals.

3. Data and Ethical Constraints

3.1. IRB status and recruitment gating

IRB approval is pending, so recruitment and human-subject data collection are currently blocked. Accordingly, this draft reports no human-subject empirical results and focuses on algorithmic methodology, implementation, and a pre-registered evaluation framework.

3.2. Planned offline evaluation design

Once IRB approval is obtained, we plan to evaluate the system using consenting established couples as offline ground truth. Each participant completes the questionnaire independently, and we measure whether the pipeline ranks their real partner highly within the cohort (Section 6). This *partner retrieval* setting is intended as a conservative sanity check that the scoring function captures some relationship signal; it is *not* a claim that an established partner is the globally optimal match, nor a proxy for relationship success.

3.3. Data artifacts and privacy

The public repository includes the questionnaire specification (`data/questions.json`) and all scoring/matching code. Evaluation scripts assume two private inputs that are *not* included in the public repo:

- `export.csv`: raw responses (contains personally identifiable information),
- `couples.csv`: mapping of consenting established couples used as offline ground truth.

Because `export.csv` contains *personally identifiable information* (PII) and sensitive preferences, it must be stored securely and should not be shared publicly. All reporting should use anonymized identifiers and avoid reproducing raw responses. Derived artifacts (scores, ranks, and aggregate tables) can be generated without copying raw PII fields.

4. Methods: Compatibility Scoring and Matching Pipeline

This section formalizes the compatibility scoring pipeline implemented in the uploaded repository snapshot. Every algorithm description below is grounded in specific file paths and function/class names.

4.1. Questionnaire representation and notation

Let $\mathscr{I} = \{1, \ldots, N\}$ be the set of participants in a cohort, with $N = |\mathscr{I}|$. Let Q be the set of question ids in the questionnaire. Each question $q \in Q$ has metadata:

- a **domain** $d(q)$ (e.g., Values, Communication),
- a **response type** (Likert or multiple-choice),
- (for Likert) a discrete scale and a user-specified distance tolerance.

A **triad response** for participant u and item q consists of:

- a **self-answer** $x_{u,q}$ (e.g., a Likert value in $\{1,2,3,4,5\}$ or a multiple-choice option; for multi-select items we treat $x_{u,q}$ as a set of chosen options),
- an **acceptable set/range** $A_{u,q}$ describing which partner answers are acceptable, and
- an **importance label** $w_{u,q}$ indicating how much u cares about this item when evaluating a partner.

In the exported participant table (the repository assumes a CSV such as `export.csv`), each participant i has:

- **self response** $x_{i,q}$ stored as a column `SELF_{qid}`,
- **acceptability specification** stored as JSON `ACC_{qid}` (or missing),
- **importance** stored as a numeric label `IMP_{qid}`.

The pipeline computes, for each ordered pair (i, j) and each item q, a *directional acceptability* value

$$a_{i\to j}(q) \in [0,1],$$

interpreted as "how acceptable is j to i on item q?"

In the deterministic baseline, $a_{i\to j}(q) \in \{0,1\}$. In PAM, $a_{i\to j}(q)$ is a probability computed by a kernel, and uncertainty is tracked.

A core design choice throughout the repository is to compute *directional* satisfaction scores first and then aggregate reciprocally.

4.2. Baseline FinalMatch (hard acceptability)

Implementation. The baseline deterministic score is implemented in:

- `algorithm/algorithms/core.py::final_match` (main scoring function),
- `algorithm/algorithms/core.py::_acceptability_ok` (item-level acceptability check),
- `algorithm/algorithms/core.py::_domain_multiplier` and `algorithm/algorithms/core.py::DEFAULT_DOMAIN_MULTIPLIER` (domain multipliers and priority boosts),
- `algorithm/algorithms/core.py::IMPORTANCE_WEIGHT` (importance mapping),
- `algorithm/algorithms/variants.py` (variant hyperparameter presets).

4.2.1. Item-level acceptability Given participants i and j and item q, the baseline computes a binary indicator

$$a_{i\to j}^{\text{hard}}(q) \in \{0,1\},$$

by calling `_acceptability_ok(a_self, a_acc, b_self, qmeta)` with $a = i$ and $b = j$.

Likert items. For Likert response types, `_acceptability_ok` parses the acceptability JSON `ACC_{qid}` (if present) to extract a tolerance `"tol"` in steps. If `tol==999`, acceptability is always true. Otherwise, it checks whether

$$|x_{i,q} - x_{j,q}| \le \tau_{i,q},$$

where $\tau_{i,q}$ is the tolerance in steps.

Multiple-choice items. For `MC_SINGLE` and `MC_MULTI`, `_acceptability_ok` constructs a set $S_i(q)$ of acceptable options: if `ACC_{qid}` is present, it uses the list under `"acc"`; otherwise it defaults to i's own selection(s) `SELF_{qid}`. Acceptability is then:

- **MC_SINGLE:** accept iff $x_{j,q} \in S_i(q)$,

- **MC_MULTI:** accept iff $S_i(q) \cap X_{j,q} \neq \emptyset$, where $X_{j,q}$ is the set of options selected by j on item q.

If an acceptability JSON is malformed, missing, or the response is missing, the function returns false or skips the item depending on the missingness checks upstream in `final_match`.

4.2.2. Importance weights, domain multipliers, and top priorities

Importance mapping. The baseline maps `IMP_{qid}` to a nonnegative weight via `IMPORTANCE_WEIGHT` in `core.py`. The default mapping is the identity on the OkCupid-style label set:

$$0 \mapsto 0, \quad 1 \mapsto 1, \quad 10 \mapsto 10, \quad 50 \mapsto 50, \quad 250 \mapsto 250.$$

This identity mapping is a tunable default; alternative mappings (e.g., compressed importance as in PAM, Section 4.3) can be substituted. This weight is applied directionally: i's importance on item q affects i's satisfaction score, and j's importance affects j's satisfaction score. These are deliberately spread out to make *Mandatory* (250) act like a near-hard constraint: one violated mandatory item can dominate many satisfied low-importance items.

Domain multipliers. Each item's contribution is multiplied by a domain multiplier. `DEFAULT_-DOMAIN_MULTIPLIER` sets:

$$\begin{aligned} \text{Values} &= 5.0, & \text{Communication} &= 3.0, \\ \text{Lifestyle} &= 3.0, & \text{Social} &= 0.5, \\ \text{Personality} &= 0.5, & \text{Friendship} &= 0.5. \end{aligned}$$

Any domain not listed defaults to 0.0 and is excluded from scoring.

Domain weighting rationale. While these multipliers are ultimately hand-tuned, their relative ordering draws on empirical findings from relationship science about which partner characteristics most strongly predict partner selection (assortative mating) and relationship outcomes.

Values receive the highest weight because partner correlations are consistently strongest for political and religious attitudes, educational attainment, and related value-laden constructs. Horwitz et al. [2023] analyzed 133 traits in the UK Biobank and reviewed meta-analyses of 22 traits, finding that partners assort most strongly on political/religious attitudes and educational attainment, while psychological and personality traits show weaker (though still positive) partner correlations. Luo and Klohnen [2005] found substantial similarity on attitude-related domains but little similarity on personality domains in newlyweds, and Gaunt [2006] demonstrated that similarity on attitudes and values is associated with marital satisfaction. More recent work by Leikas et al. [2018] using response-surface methods confirms that attitudes and values constitute a distinct similarity domain with measurable links to relationship satisfaction. A meta-analysis by Montoya et al. [2008] further supports the general premise that similarity, especially in attitudes and values, is tied to attraction and relationship formation.

Communication receives high weight because conflict communication patterns show robust associations with relationship outcomes and stability. Schrodt et al. [2014] conducted a meta-analysis of the demand/withdraw interaction pattern and reported a moderate association ($r \approx .36$) between this pattern and relationship outcomes. The classic longitudinal review by Karney and Bradbury [1995] established that interaction patterns, particularly negative affect during conflict, are key predictors of marital quality and stability. Gottman and Levenson [1992] provided observational evidence that specific communication behaviors during conflict predict later dissolution. Although Lavner et al. [2016] note that the causal direction between communication and satisfaction is complex, the cross-sectional associations are strong, justifying communication as a high-leverage domain.

Personality similarity receives lower weight for two reasons: (i) partner similarity on Big Five traits is empirically small, and (ii) personality effects on relationship outcomes operate primarily through actor/partner trait levels rather than similarity. Weidmann et al. [2023] found that personality similarity often plays a negligible role in explaining relationship satisfaction across both traits and facets. Bach and Malouff [2025] reports partner-trait correlations for Big Five in the range $0.00 \leq r \leq 0.15$, indicating low resemblance. While Malouff et al. [2010] found that Big Five traits do relate to relationship satisfaction (with neuroticism showing the strongest negative association), these effects are actor/partner effects rather than similarity effects. This suggests that personality is relevant but should not dominate scoring; the current implementation captures personality through trait-distance features rather than relying heavily on personality similarity in the acceptability weighting.

Lifestyle receives moderate weight because many day-to-day traits and behaviors show substantial partner assortment. Horwitz et al. [2023] found meaningful partner correlations on lifestyle-related traits including substance use behaviors and daily habits. These domains plausibly affect friction and fit in daily life, justifying a weight above personality similarity but below values and communication.

Measurement considerations. The relatively low weight on Personality also reflects measurement limitations. The questionnaire uses short-form personality items inspired by the Ten-Item Personality Inventory [TIPI; Gosling et al., 2003], which trades reliability for brevity. In small samples, short personality measures introduce substantial measurement error, making it prudent to downweight personality similarity in the absence of more reliable assessment. The Values domain draws on item structures inspired by the Portrait Values Questionnaire [Schwartz, 2012, Schwartz et al., 2001], which has stronger psychometric foundations and supports the heavier weighting.

Epistemic humility. We emphasize that compatibility claims in online dating require humility. As Finkel et al. [2012] argue, many compatibility and matching claims lack strong evidence, and transparent evaluation is essential. The present weighting scheme is a starting point informed by existing literature, not a claim of optimality. The modular design allows weights to be adjusted as empirical evidence accumulates from actual deployments.

Top-3 priorities boost. Participants also answer a priority question (qid `47`) used to extract a set of top priorities. `core.py::extract_priorities_from_q47` parses `SELF_47` as JSON and returns a list of selected domains; we denote the resulting set by P_i for participant i. `core.py::_domain_multiplier` multiplies the base domain multiplier by a `priority_boost` (default 1.5 in `final_match`) when the item's domain is in the participant's extracted priorities.

4.2.3. Directional satisfaction and reciprocal aggregation For a pair (i, j), `final_match` iterates over all questions in `qmeta["questions"]` and accumulates weighted acceptability for each direction.

Define the directional item weight:

$$w_{i,q} = \texttt{IMPORTANCE\_WEIGHT}(I_{i,q}) \cdot \texttt{\_domain\_multiplier}(d(q); \text{priorities}_i),$$

where $I_{i,q}$ is the raw importance label for participant i on item q, and priorities_i is the set of top-3 priority domains extracted from participant i's response to question 47. The `_domain_multiplier` function returns the base domain multiplier (from `DEFAULT_DOMAIN_MULTIPLIER`) scaled by 1.5 if $d(q)$ is in the participant's priority set.

An item q contributes only if at least one direction has nonzero weight (the implementation skips only when *both* $w_{i,q} = 0$ and $w_{j,q} = 0$). For each contributing item, it computes $a^{\text{hard}}_{i\to j}(q)$ and $a^{\text{hard}}_{j\to i}(q)$ and accumulates:

$$\text{num}_{i\to j} \leftarrow \text{num}_{i\to j} + w_{i,q}\, a^{\text{hard}}_{i\to j}(q), \qquad \text{den}_{i\to j} \leftarrow \text{den}_{i\to j} + w_{i,q},$$

and similarly for $j \to i$.

Directional satisfaction (as implemented) is:

$$s_i(j) = \frac{\text{num}_{i\to j}}{\text{den}_{i\to j}} \in [0,1], \tag{1}$$

with the convention that if $\text{den}_{i\to j} = 0$ then $s_i(j) = 0$ (see the `if denA > 0 else 0.0` logic in `final_match`).

Mutual aggregation. The baseline then aggregates the two directional satisfactions into a symmetric match score. If `use_symmetric_mean=False` (the default in `final_match`), it uses a geometric mean:

$$\text{match}(i,j) = 100 \cdot \sqrt{s_i(j)\, s_j(i)}. \tag{2}$$

If `use_symmetric_mean=True`, it uses an arithmetic mean (note: both the geometric and arithmetic means are symmetric functions; the parameter name reflects the implementation convention):

$$\text{match}(i,j) = 100 \cdot \frac{s_i(j) + s_j(i)}{2}.$$

4.2.4. Overlap filter and finite-sample penalty

Overlap counting. `final_match` counts the number of items n that were considered (incremented once per question when at least one side's weight is nonzero and neither self response is missing). If $n < n_{\min}$ (parameter `min_overlap`, default 20), it returns `None`.

Finite-overlap penalty. If $n \geq n_{\min}$, it applies a penalty of the form $c/\sqrt{n}$, implemented as:

$$\text{penalty}(n) = \frac{c}{\sqrt{n}},$$

where c is `c_penalty` (default 100.0). The final score is:

$$\text{final}(i,j) = \max\{0,\ \text{match}(i,j) - c/\sqrt{n}\}. \tag{3}$$

Hyperparameter variants. `algorithm/algorithms/variants.py` defines named variants such as `core_c100_n20_geo` (geometric mean, $c = 100$, $n_{\min} = 20$) and `symm_c100_n20_mean` (arithmetic mean, $c = 100$, $n_{\min} = 20$), and additional variants over $c \in \{60, 100, 140\}$ and $n_{\min} \in \{15, 20, 25\}$ intended for ablation studies.

4.3. Probabilistic Acceptability Model (PAM) + LCB

Implementation. PAM and its uncertainty-aware LCB scoring are implemented across:

- Acceptability kernels: `algorithm/models/accept_kernels.py::AcceptabilityParams`, `p_likert`, `p_mc`,
- Importance compression: `algorithm/models/importance_map.py::COMPRESSED_IMPORTANCE`,
- Domain weights: `algorithm/models/domain_weights.py::DEFAULT_DOMAIN_WEIGHTS`,

- Multiple-choice similarity: `algorithm/models/mc_similarity.py::build_mc_similarity`,
- Directional stats (mean/variance):
  `algorithm/scoring/directional.py::directional_stats`,
- Lower confidence bounds: `algorithm/eval/evaluate_pam.py::lcb_from`,
- Symmetric pair scoring: `algorithm/scoring/pair_score.py::score_pair` and
  `algorithm/scoring/pair_score.py::combine_scores`.

Hard acceptability can be brittle: it assigns the same penalty to slightly-off and wildly-off answers, and it ignores the uncertainty introduced by coarse tolerance categories. PAM replaces the baseline's hard acceptability $a^{\text{hard}}_{i\to j}(q)$ with a soft probability

$$p_{i\to j}(q) \in [0,1],$$

and tracks uncertainty through an estimated variance of the weighted mean.

4.3.1. Likert acceptability kernel For Likert items, `accept_kernels.py::p_likert` maps an absolute step distance Δ and a tolerance in steps τ to a probability.

The function computes:

$$x = \frac{\Delta}{\tau+1},$$

and then applies a logistic transform with a piecewise offset based on τ:

$$p_{\text{likert}}(\Delta,\tau) = \frac{1}{1+\exp(\alpha(x-\eta_\tau(x)))},$$

where $\alpha =$ `AcceptabilityParams.alpha` and $\eta_\tau(x)$ is either b_τ or a_τ depending on whether $x < b_\tau$ (see the piecewise `if x < b:  z = alpha*(x-a)` logic). The default parameters are provided by `AcceptabilityParams.from_defaults`:

$$\alpha = 5.0, \quad a_\tau \in \{-1.5,-1.0,-0.7\} \text{ for } \tau \in \{0,1,2\}, \quad b_\tau \in \{0.8,0.6,0.4\} \text{ for } \tau \in \{0,1,2\}.$$

(Exact values are in `accept_kernels.py::from_defaults`.)

4.3.2. Multiple-choice kernel For multiple-choice items, `directional.py::directional_-stats` computes a similarity score between j's selection(s) and i's acceptable set by taking a maximum similarity over option pairs using a precomputed similarity map (from `mc_similarity.py`). It then applies `accept_kernels.py::p_mc(sim)`:

$$p_{\mathrm{mc}}(s) = \frac{1}{1+\exp(\alpha_{\mathrm{mc}}(1-s))},$$

where $\alpha_{\mathrm{mc}} =$ `AcceptabilityParams.alpha_mc` (default -1.0). Because $\alpha_{\mathrm{mc}} < 0$ by default, p_{mc} increases as similarity s increases.

4.3.3. Compressed importance mapping Unlike the baseline, PAM compresses the importance weights so that "mandatory" items don't dominate. `importance_map.py::COMPRESSED_-IMPORTANCE` maps:

$$0 \mapsto 0, \quad 1 \mapsto 1, \quad 10 \mapsto 3, \quad 50 \mapsto 6, \quad 250 \mapsto 10.$$

The rationale is that probabilistic acceptability already introduces smoothness; extreme weight ratios can destabilize variance estimates and confidence bounds.

Directional item weights in PAM are then:

$$w_{i,q}^{\mathrm{PAM}} = \mathrm{CompImp}(I_{i,q}) \cdot W_{\mathrm{dom}}(d(q)),$$

where W_{dom} is a domain weight (Section 4.3.4).

4.3.4. Domain weights PAM uses the same base domain multipliers as the baseline via `domain_-weights.py::DomainWeights.from_defaults`, which wraps `core.py::DEFAULT_DOMAIN_MULTIPLIER`. In the default map:

$$\begin{aligned} \text{Values} &= 5.0, & \text{Communication} &= 3.0, \\ \text{Lifestyle} &= 3.0, & \text{Social} &= 0.5, \\ \text{Personality} &= 0.5, & \text{Friendship} &= 0.5. \end{aligned}$$

Unlisted domains default to 0.0 and are excluded from scoring. These weights are applied multiplicatively to compressed importance. The rationale for this weighting hierarchy (Values > Communication/Lifestyle > Personality/Social/Friendship) is provided in Section 4.2.2 and draws on assortative mating and relationship outcome research [Horwitz et al., 2023, Luo and Klohnen, 2005, Schrodt et al., 2014, Weidmann et al., 2023].

4.3.5. Directional mean/variance and LCB

Directional weighted mean. For a fixed ordered pair (i, j), `directional_stats` computes:

$$\hat{s}_i(j) = \frac{\sum_{q \in Q_{ij}} w_{i,q}^{\text{PAM}} \, p_{i \to j}(q)}{\sum_{q \in Q_{ij}} w_{i,q}^{\text{PAM}}},$$

where Q_{ij} are questions that have kernel parameters, nonzero weight, and defined self-responses.

Directional variance proxy. The same function computes an estimated variance of the weighted mean using a Bernoulli proxy:

$$\widehat{\text{Var}}[\hat{s}_i(j)] = \frac{\sum_{q \in Q_{ij}} (w_{i,q}^{\text{PAM}})^2 \, p_{i \to j}(q) \, (1 - p_{i \to j}(q))}{\left(\sum_{q \in Q_{ij}} w_{i,q}^{\text{PAM}}\right)^2}.$$

It also tracks an "effective sample size" proxy:

$$n_{\text{eff}} = \frac{\left(\sum_q w_q\right)^2}{\sum_q w_q^2},$$

implemented as `neff = (Wsum**2)/(W2sum + 1e-12)`.

Lower confidence bound. Given $(\hat{s}, \widehat{\mathrm{Var}}, n_{\mathrm{eff}})$, `evaluate_pam.py::lcb_from` computes a lower confidence bound. The default is `kind="normal"`, which uses:

$$\mathrm{LCB} = \max\{0,\ \hat{s} - z\sqrt{\widehat{\mathrm{Var}}}\},$$

where $z = \Phi^{-1}(1-\delta)$ and δ is a configurable tail probability (default 0.1 in `lcb_from`). The function also supports `kind="bernstein"` (empirical Bernstein-style) and `kind="hoeffding"`.

4.3.6. Symmetric combination rule Given directional LCBs for both directions, PAM forms a symmetric score by combining them in log-odds space. `pair_score.py::combine_scores` defines:

$$\mathrm{score}(i,j) = \mathrm{logit}(\mathrm{LCB}_{i\to j}) + \mathrm{logit}(\mathrm{LCB}_{j\to i}),$$

where `logit` clips inputs to $[\varepsilon, 1-\varepsilon]$ for numerical stability.

This rule penalizes pairs where either direction has a small LCB (since $\mathrm{logit}(p) \to -\infty$ as $p \to 0$). It explicitly enforces reciprocity: both directional acceptability estimates matter.

4.3.7. End-to-end PAM scoring `pair_score.py::score_pair` ties the above pieces together: it calls `directional_stats` twice (once for $i \to j$ and once for $j \to i$), computes LCBs and radii (confidence radii), and returns both the combined symmetric score and a diagnostic dictionary.

A separate evaluation script `algorithm/eval/evaluate_pam.py::main` loads participant data, computes PAM scores for true couples and sampled non-partners, and writes a summary CSV (default `algorithm/out/summary_pam.csv`) plus a per-couple pairs file with the same basename and suffix `.pairs.csv` .

4.4. Distance-kernel models: per-domain mutuality and trait distance

High-level idea. A separate family of models decomposes compatibility into domain-wise components and explicitly allows each domain to behave as either a *similarity* domain (partners closer in traits are better) or a *complementarity* domain (partners farther apart are better), with an *irrelevant* option. The models operate on precomputed feature tensors:

- **Mutual acceptability tensor** $m_{ij}^{(d)}$ derived from directional LCBs for domain d,
- **Distance tensor** $\delta_{ij}^{(d)}$ measuring how far apart two participants are in that domain (from normalized self-responses),
- **Reliability tensor** $r_{ij}^{(d)}$ (currently set to 1 everywhere in `make_features.py`).

Implementation reference.

- Pairwise tensor construction: `algorithm/features/build_pair_tensors.py::build_lcbs`, `algorithm/features/build_pair_tensors.py::build_distances`, and `algorithm/features/build_pair_tensors.py::main`.
- Feature packaging: `algorithm/features/make_features.py::main`.
- Configurable scoring: `algorithm/scoring/config_scorer.py::ConfigurableScorer` and `algorithm/scoring/config_scorer.py::score_pair`.
- Learning: `algorithm/ml/soft_gated_trainer.py::SoftGatedTrainer` and `algorithm/ml/evolutionary_trainer.py::evolutionary_search`.

4.4.1. Feature tensors

Directional LCB tensors. `build_pair_tensors.py::build_lcbs` computes, for each ordered pair (i, j) and each domain d:

$$\mathrm{LCB}_{i\to j}^{(d)} \in [0,1], \qquad r_{i\to j}^{(d)} \geq 0, \qquad n_{\mathrm{eff},i\to j}^{(d)} \geq 0,$$

and returns them as numpy arrays with shape `(N, N, D, 2)`, where the final axis indexes direction: index 0 stores $i \to j$ and index 1 stores $j \to i$ for the same ordered pair `(i,j)`. (See the assignments in the nested loops of `build_lcbs`.)

The routine internally uses the PAM directional scoring functions: it builds a per-domain list of question ids, computes directional means and variances using `algorithm/scoring/directional.py::directional_stats`, and converts them into LCBs via `algorithm/eval/evaluate_pam.py::lcb_from`.

Mutual acceptability. Given the directional LCB tensor, `make_features.py::main` constructs a mutual acceptability tensor:

$$m_{ij}^{(d)} = \sqrt{\mathrm{LCB}_{i\to j}^{(d)} \cdot \mathrm{LCB}_{j\to i}^{(d)}},$$

implemented as `mutual = np.sqrt(lcb[:,:,:,0] * lcb[:,:,:,1])`.

Trait distances. `build_pair_tensors.py::build_distances` constructs a per-domain distance tensor from self-responses as follows.

First, it loads a normalization JSON (produced by `algorithm/preprocess/normalize.py`) that stores per-item min and max values. For each domain d, it creates a matrix $X^{(d)} \in \mathbb{R}^{N\times|Q_d|}$ of normalized self-responses:

$$X_{i,q}^{(d)} = \mathrm{clip}_{[0,1]}\left(\frac{x_{i,q} - \min_q}{(\max_q - \min_q) + 10^{-9}}\right),$$

for each question id q in domain d that appears in the normalization JSON. Entries are set to `NaN` if the self-response is missing or non-numeric.

Then, for each pair (i, j), it computes the mean absolute difference over questions where both normalized values are present:

$$\delta_{ij}^{(d)} = \frac{1}{|Q_{ij}^{(d)}|} \sum_{q\in Q_{ij}^{(d)}} \left|X_{i,q}^{(d)} - X_{j,q}^{(d)}\right|,$$

where $Q_{ij}^{(d)}$ is the set of questions in domain d for which both $X_{i,q}^{(d)}$ and $X_{j,q}^{(d)}$ are finite. This is implemented by broadcasting differences and averaging only over finite entries (see `dom_dist = sum_diff / cnt` in `build_distances`). If $|Q_{ij}^{(d)}| = 0$, the implementation leaves $\delta_{ij}^{(d)}$ at its initialized value of 0.0 for that domain/pair.

Reliability. In the packaged features file, `make_features.py` currently sets the reliability tensor to all ones: `rel = np.ones_like(dist, dtype=np.float32)`. This is a placeholder for later extensions where domains with few contributing items might be downweighted.

Kernel grids. `make_features.py::main` also writes fixed candidate grids: `mus_sims = [0.0, 0.25]`, `mus_comps = [0.5, 0.75, 1.0]`, and `sigmas = [0.15, 0.25, 0.35]` (all float32), which are used by the evolutionary configuration search.

4.4.2. Configurable scoring function Given tensors $(m_{ij}^{(d)}, \delta_{ij}^{(d)}, r_{ij}^{(d)})$ and a per-domain configuration, `ConfigurableScorer.score_pair` computes:

$$S(i,j) \; = \; \sum_{d} w_d \cdot \big(m_{ij}^{(d)}\big)^{\alpha_d} \cdot \big(k_d(\delta_{ij}^{(d)})\big)^{\beta_d} \cdot r_{ij}^{(d)}, \tag{4}$$

with nonnegative weights w_d and exponents α_d, β_d.

The kernel k_d is chosen according to `mode`:

$$k_{\text{sim}}(\delta;\sigma) = \exp\left(-\left(\frac{\delta}{\sigma}\right)^2\right), \qquad k_{\text{comp}}(\delta;\mu,\sigma) = \exp\left(-\frac{1}{2}\left(\frac{\delta-\mu}{\sigma}\right)^2\right).$$

Here $\delta = \delta_{ij}^{(d)} \in [0,1]$ is the normalized mean absolute trait distance in domain d (Section 4.4). The similarity kernel k_{sim} is an RBF/Gaussian centered at $\delta = 0$; the bandwidth $\sigma_d > 0$ controls how quickly similarity decays as two participants differ. The complementarity kernel k_{comp} is a Gaussian centered at $\delta = \mu_d$, where $\mu_d \in [0,1]$ specifies the *target distance* (e.g., "moderately different") and σ_d specifies tolerance around that target.

- **similarity:** $k_d(\delta) = k_{\text{sim}}(\delta;\sigma_d)$,
- **complementarity:** $k_d(\delta) = k_{\text{comp}}(\delta;\mu_d,\sigma_d)$,
- **mixed:** $k_d(\delta) = \frac{1}{2}\big(k_{\text{sim}}(\delta;\sigma_d) + k_{\text{comp}}(\delta;\mu_d,\sigma_d)\big)$,
- **irrelevant:** $k_d(\delta) = 1$.

(See `k_sim`, `k_comp`, and the mode logic in `config_scorer.py::score_pair`.)

Kernel intuition.

- The *similarity kernel* k_{sim} peaks at $\delta = 0$ (identical trait values) and decays as distance increases. Smaller σ makes it more selective (sharply penalizing even small differences).

- The *complementarity kernel* k_{comp} peaks at $\delta = \mu$ and penalizes both distances below and above the target. This models the hypothesis that moderate differences (e.g., $\mu = 0.5$) are optimal for some traits. The bandwidth σ controls how sharply the kernel falls off around the optimal difference.

4.4.3. Soft-gated training

Model. The soft-gated model learns parameters per domain d: a nonnegative domain weight w_d (implemented as `relu(self.w[d])`), a gating intercept/slope (a_d, b_d), and kernel parameters (μ_d, σ_d). For a given pair (i, j) in domain d, it computes a gate value

$$g_{ij}^{(d)} \;=\; \text{sigmoid}\big(a_d + b_d\,\delta_{ij}^{(d)}\big),$$

where $\text{sigmoid}(t) = \frac{1}{1+\exp(-t)}$.

and combines similarity and complementarity kernels *per pair*:

$$k_{ij}^{(d)} \;=\; g_{ij}^{(d)}\,k_{\text{comp}}(\delta_{ij}^{(d)}; \mu_d, \sigma_d) + (1 - g_{ij}^{(d)})\,k_{\text{sim}}(\delta_{ij}^{(d)}; \sigma_d).$$

The final score is:

$$S_{\text{soft}}(i, j) \;=\; \sum_d \text{ReLU}(w_d) \cdot m_{ij}^{(d)} \cdot k_{ij}^{(d)},$$

matching `SoftGatedTrainer.score`.

Objective. Training uses a *Bayesian Personalized Ranking* (BPR)-style pairwise loss over triples $(i, p(i), n)$ where $p(i)$ is the true partner and n is a sampled negative:

$$\mathscr{L}_{\text{BPR}}(\theta) \;=\; -\log \text{sigmoid}\big(S_\theta(i, p(i)) - S_\theta(i, n)\big),$$

implemented in `SoftGatedTrainer.step` via a logistic loss. This pushes the model to score true partners above sampled non-partners.

Regularization. The implementation adds: (i) optional ℓ_1 penalty on the ReLU weights w_d (`l1`), (ii) an "exclusivity" penalty `relu(a)*relu(b)` averaged over domains (`excl`), and (iii) a penalty that discourages very small σ (`sigma_penalty` in `step`).

Interpretable mode labeling. After training, `SoftGatedTrainer.infer_modes` produces a domain-level label by comparing the magnitudes of $\mathrm{ReLU}(a_d)$ and $\mathrm{ReLU}(b_d)$: if w_d is tiny the domain is labeled irrelevant; if $\mathrm{ReLU}(a_d) \geq 2\mathrm{ReLU}(b_d)$ it's labeled similarity; if $\mathrm{ReLU}(b_d) \geq 2\mathrm{ReLU}(a_d)$ it's labeled complementarity; otherwise it's labeled mixed.

4.4.4. Evolutionary configuration search The evolutionary search procedure (`algorithm/ml/evolutionary_trainer.py::evolutionary_search`) performs a discrete random search over domain configurations. A configuration specifies, per domain, a mode (similarity/complementarity/irrelevant), weights, and kernel parameters (μ, σ) drawn from the candidate grids in `features.npz`. Given a configuration, scoring is performed by `ConfigurableScorer`, and metrics are computed using the evaluation harness `algorithm/eval/harness.py::run_harness`.

The objective used to select the best configuration is defined in `evolutionary_search` as:

$$J = 3 \cdot \mathrm{Hit@1} + 2 \cdot \mathrm{Mutual@K} + 1 \cdot \mathrm{Hit@K} + 0.5 \cdot \mathrm{AUC},$$

where K is the harness top-K parameter (default `topk=3`).

Objective term rationale.

- **Hit@1** (weight 3): Measures whether the true partner is ranked first. This receives the highest weight because exact top-1 recovery is the most demanding test of compatibility scoring.
- **Mutual@K** (weight 2): Measures whether both partners in a couple rank each other in the top-K. This captures reciprocal success, which is essential for matching applications where both sides must find each other acceptable.
- **Hit@K** (weight 1): Measures whether the true partner appears anywhere in the top-K list. This is a softer criterion than Hit@1 and rewards configurations that place partners near the top even if

not exactly first.

- **AUC** (weight 0.5): Measures overall separation between true-couple scores and random-pair scores. This receives the lowest weight because it captures global ranking quality rather than top-K precision, which is more relevant for recommendation.

In the evaluation plan, this objective is used only for *model selection* inside the evolutionary search; final reporting uses the full metric suite on held-out data.

4.5. Merged IDF+LCB pipeline and matching

Implementation. The merged pipeline and matching solvers referenced in this report are:

- IDF weighting and merged scoring: `algorithm/matching/merged_pipeline.py::compute_-idf_weights`, `algorithm/matching/merged_pipeline.py::score_pair_idf_lcb`, `algorithm/matching/merged_pipeline.py::build_preference_lists`, and `algorithm/matching/merged_pipeline.py::summarize_true_partner_ranks`.
- Deferred acceptance (DA): `algorithm/matching/da.py::gale_shapley_da` and `algorithm/matching/da.py::da_with_ties`.
- Stable roommates: `algorithm/matching/irving.py::stable_roommates`.
- Maximum-weight matching: `algorithm/matching/blossom.py::max_weight_matching`.
- "Best stable" LP/MILP: `algorithm/algorithms/best_stable_lp.py::best_stable_matching`.

4.5.1. IDF-style item reweighting

The merged pipeline introduces an IDF-style reweighting to emphasize "rarer" traits/items. It builds per-item inverse-frequency weights from item similarity statistics computed over the cohort.

Item similarity. For each question q, `merged_pipeline.py::compute_idf_weights` constructs an `ItemSim` object using `build_item_sims`.

- For Likert items, similarity is defined as $1-\Delta/(S-1)$, where Δ is the absolute difference in step values and S is the number of steps (see `item_sims.py::LikertSim.sim`).
- For MC_SINGLE, similarity is 1 if equal else 0.

- For MC_MULTI, similarity is the Jaccard index between the two option sets.

IDF transform with shrinkage. For each item, it computes an average similarity across all ordered pairs $i \neq j$ and applies an IDF transform. `rarity.py::idf_with_shrink` first shrinks the observed average similarity toward a prior mean using a Beta prior parameterized by `prior_-strength` and `prior_mean` (see `idf_with_shrink` signature), and then returns:

$$\mathrm{idf}(q) = \log\left(\frac{1}{\max(\varepsilon,\, \bar{s}_q)}\right),$$

where $\bar{s}_q$ is the (shrunk) mean similarity and ε is a small clip constant. This yields larger weights for items whose average similarity is low (i.e., responses are more "specific" in this cohort).

4.5.2. Soft-capped domain contributions `merged_pipeline.py::score_pair_idf_lcb` computes a reciprocal score by aggregating domain-level components and applying a soft cap to avoid single-domain domination.

For each domain d, it computes directional LCBs and radii using `directional_stats` and `lcb_from`, but with per-item weights multiplied by IDF and by a base importance weight `w_base`. It then forms a mutual domain score (geometric mean) and aggregates across domains. Before final aggregation, it applies a soft cap parameter `softcap_rho`: if the top domain contributes more than a ρ fraction of the total, it rescales the top contribution downward (see the explicit logic in `score_pair_idf_lcb`).

4.5.3. Preference tiers with ties Given a score (and an uncertainty radius) for each candidate partner, `merged_pipeline.py::build_preference_lists` constructs preference tiers rather than strict rankings.

For each participant i, it sorts candidates j by decreasing LCB score. It then groups consecutive candidates into the same tier if the LCB gap is not large relative to uncertainty: if the previous candidate had LCB ℓ_{prev} with radius r_{prev} and the current candidate has LCB ℓ with radius r, they are tied if

$$\ell_{\text{prev}} - \ell \;\leq\; \tau \cdot (r_{\text{prev}} + r),$$

where `tie_tau` is a configurable parameter (default 0.5). This produces a list of tiers `tiers[i]` where each tier is a list of candidate indices.

The same routine can enforce an acceptability cutoff (drop candidates whose LCB is below `min_lcb`) and can apply an additional mutuality boost for pairs who appear in each other's top-K tiers (parameter `mutual_boost`).

4.5.4. From preference tiers to matchings The repository supports multiple matching solvers that consume preference information:

Deferred acceptance (DA). In a bipartite setting, `matching/da.py::gale_shapley_da` runs the standard proposer-optimal DA algorithm. The repository also includes a tie-aware version `da_with_ties` that breaks ties randomly by shuffling within tiers (see `da.py`).

Stable roommates. For a one-population setting with strict preferences, `matching/irving.py::stable_roommates` implements Irving's algorithm.

Max-weight matching (blossom). Given a symmetric weight matrix, `matching/blossom.py::max_weight_matching` uses NetworkX's max-weight matching to produce a maximum-weight pairing (not necessarily stable).

Best stable LP/MILP. `algorithm/algorithms/best_stable_lp.py::best_stable_matching` formulates an optimization problem over matchings that are stable with respect to provided preference lists. It constructs a weight matrix from scores, introduces binary decision variables for matches, adds feasibility constraints (each participant matched to at most one partner), and then adds stability constraints based on preference ranks (see the construction of `better_or_equal` sets and constraints in `best_stable_lp.py`). It then solves via PuLP and returns the best stable matching under the objective. If the solver does not report an optimal solution, it falls back to a DA-based matching (see the `if LpStatus... != "Optimal"` branch).

5. Implementation Details

This section documents how the repository is organized and how the evaluation scripts connect.

5.1. Code organization and responsibilities

The repository is organized as:

- **Baseline compatibility:** `algorithm/algorithms/core.py` and `algorithm/algorithms/variants.py`.
- **PAM model components:** `algorithm/models/accept_kernels.py`, `algorithm/models/importance_map.py`, `algorithm/models/domain_weights.py`, `algorithm/models/mc_similarity.py`.
- **Directional and pair scoring:** `algorithm/scoring/directional.py`, `algorithm/scoring/pair_score.py`.
- **Feature tensors and ML:** `algorithm/features/*`, `algorithm/ml/soft_gated_trainer.py`, `algorithm/ml/evolutionary_trainer.py`, `algorithm/scoring/config_scorer.py`.
- **Matching solvers and merged pipeline:** `algorithm/matching/*`, `algorithm/algorithms/best_stable_lp.py`.
- **Evaluation harness and metrics:** `algorithm/eval/harness.py`, `algorithm/eval/evaluate_couples.py`, and scripts under `algorithm/paper/` used to compare algorithms and generate report tables.

5.2. Evaluation scripts and outputs

Given private inputs (`export.csv`, `couples.csv`) after IRB approval, the evaluation scripts generate outputs such as:

- `algorithm/out/` and `algorithm/out_ml/` (baseline and baseline+learned weights),
- `algorithm/out_pam/` (PAM training and evaluation artifacts),
- `algorithm/out_soft/` (soft-gated training),
- `algorithm/out_evo/` (evolutionary search),
- `algorithm/out_merged/` (merged IDF+LCB pipeline and preference tiers),
- `algorithm/out_paper/` (cross-algorithm comparison tables for the report).

6. Evaluation Framework

This section specifies how we will evaluate scoring models and matching solvers once IRB-approved data collection begins. The evaluation code paths referenced throughout the report (`algorithm/eval/*` and `algorithm/paper/*`) already compute the metrics below; the goal here is to define the protocol and clarify what is being compared.

6.1. Offline evaluation setting

We evaluate in a closed cohort $\mathscr{I} = \{1, \dots, N\}$ of participants who complete the questionnaire. For offline ground truth, we plan to recruit consenting established couples and record a partner mapping $p(i)$ for each i (used *only* for evaluation).

For a chosen scoring model, we compute a pairwise score matrix $S(i, j)$ for all $i \neq j$ (and, when available, uncertainty diagnostics such as confidence radii). This score matrix induces a per-person ranking of candidates for each anchor i by sorting $S(i, j)$ in descending order. Separately, matching solvers consume either (i) these scores directly or (ii) derived preference lists/tiers to produce a cohort-level matching outcome M.

6.2. Metrics

The planned evaluation reports three families of metrics: (i) rank-based partner retrieval, (ii) true-vs-random separation, and (iii) matching-level stability/quality diagnostics.

6.2.1. Rank-based partner retrieval metrics For each anchor participant i, let rank_i be the 1-indexed position of their true partner $p(i)$ in the sorted list of candidates (rank 1 is best). We report:

- **Hit@K:** $\frac{1}{N} \sum_i \mathbf{1}\{\text{rank}_i \leq K\}$.
- **MRR:** $\frac{1}{N} \sum_i \frac{1}{\text{rank}_i}$.
- **Median/mean rank:** the median and mean of $\{\text{rank}_i\}_{i=1}^{N}$.
- **Mutual@K:** the fraction of *couples* (a, b) such that a ranks b in top K and b ranks a in top K.

These metrics evaluate the pairwise scoring function as a reciprocal ranking model.

6.2.2. True-vs-random separation metrics (AUC and lift) In addition to within-cohort ranks, we evaluate whether true-couple scores are separated from non-partner scores. For each couple (a,b), we sample non-partners (e.g., b' drawn uniformly from $\mathscr{I} \setminus \{a,b\}$) and compare the distribution of true scores $S(a,b)$ against sampled non-partner scores $S(a,b')$. We report:

- **AUC_std:** the probability that a randomly chosen true score exceeds a randomly chosen sampled non-partner score (ties count as 0.5), i.e., the usual Mann–Whitney AUC convention Hanley and McNeil [1982], Mann and Whitney [1947].
- **Lift:** a scale-dependent diagnostic comparing the mean true score to the mean sampled non-partner score (exact definition is reported by `evaluate_couples.py`).

Repository AUC convention. Some repository scripts compute an inverted AUC due to rank-order conventions. To avoid ambiguity, we will report both the raw repository quantity (when applicable) and the standard AUC above, using the relation AUC_std $= 1 -$ AUC_repo when the inversion applies.

6.2.3. Matching-level diagnostics When a matching solver produces a cohort-level matching M, we additionally report:

- **Match rate:** fraction of participants matched (or fraction left unmatched).
- **Total weight:** $\sum_{(i,j)\in M} w_{ij}$ for a chosen symmetric weight matrix (typically derived from $S(i,j)$).
- **Blocking pairs:** number (or fraction) of blocking pairs under the induced preference lists. For a pair (i,j), (i,j) is blocking if both i and j strictly prefer each other to their assigned outcome (or to being unmatched).

These diagnostics help distinguish *good rankings* from *good global outcomes*, and quantify trade-offs between stability and weight optimality.

6.3. Model selection and train/test protocol

Learned models (soft-gated training, evolutionary search, and any trained PAM variants) require a train/test protocol to avoid overfitting. Our default plan is *couple-wise* cross-validation: split couples into folds, train on the participants in training couples, select hyperparameters on a validation fold, and report metrics on held-out couples. This prevents information leakage where one member of a couple appears in training and the other in testing.

For evolutionary search, the objective in Section 4.4.4 is used only for selecting configurations on training/validation data; all reported metrics use held-out data. For soft-gated training, we will similarly tune regularization and optimizer hyperparameters via validation.

6.4. Planned analyses and diagnostics

Beyond aggregate metrics, we plan to report:

- **Ablations:** sensitivity to overlap thresholds, domain multipliers, priority boosts, and uncertainty parameters (e.g., LCB confidence level).
- **Per-domain contributions:** how much each domain contributes to $S(i, j)$ for true couples vs non-partners.
- **Failure-mode analysis:** cases where $p(i)$ ranks poorly, decomposed by missingness, strictness of acceptability, and high-uncertainty domains.
- **Uncertainty calibration:** whether low-LCB pairs correspond to higher observed ranking error.

Statistical uncertainty in metrics will be summarized with bootstrap confidence intervals over couples/participants and, where appropriate, permutation tests for paired comparisons between models.

7. Limitations and Ethical Considerations

7.1. Data scarcity and overfitting risk

IRB approval is pending, so this draft reports no human-subject empirical results and no statistically grounded model comparisons. Even after approval, early cohorts may be small, which has several implications:

- **Low statistical power:** small cohorts make it difficult to distinguish models whose performance differs modestly.
- **Overfitting risk:** learned models (soft-gated, trained PAM variants, evolutionary selection) can

fit idiosyncrasies of a cohort rather than general signal.

- **High metric variance:** rank-based metrics such as Hit@K can change materially when a single participant's partner rank changes.
- **Selection bias:** established couples used for offline ground truth may not represent early-stage dating preferences or the broader target population.

The pipeline includes several design choices intended to mitigate these risks: uncertainty-aware scoring (LCBs), explicit regularization, couple-wise cross-validation, and reporting of confidence intervals and ablations (Section 6).

7.2. IRB gating and deployment constraints

IRB approval is pending. Until approval, the questionnaire cannot gather any participants or be deployed for broad recruitment, and algorithm development must be framed as offline prototyping and pipeline validation. This report therefore does not propose or claim any real-world effect sizes or deployment success.

7.3. Consent, privacy, and data minimization

The raw dataset contains sensitive information (demographics, personal preferences) and PII. Any real deployment must ensure informed consent, clear communication of data use, and the ability to withdraw. A conservative default is data minimization: collect only what is needed for matching, store it securely, and avoid publishing raw responses.

In this project, `export.csv` is treated as private. Derived artifacts (scores, ranks, and aggregate tables) are generated without copying raw PII fields.

7.4. Transparency and the risk of over-exposure

Transparency is a design goal: participants should understand how matches are produced. However, transparency can also create risks. If explanations reveal too much about a participant's acceptability constraints, others might infer sensitive attributes (e.g., religion, political views) even without explicit disclosure. A practical explainability layer should therefore:

- explain *categories* (“we matched on values and communication”) rather than raw answers,
- allow users to opt out of showing certain explanations, and
- avoid exposing another participant’s declared constraints.

7.5. Bias, fairness, and social harms

Dating and friendship matching systems can encode and amplify bias. If certain preferences correlate with protected attributes, then respecting preferences can still create disparate outcomes. In a small campus community, harm can also be social rather than statistical: embarrassment, conflict, or harassment.

More broadly, compatibility claims in online dating require epistemic humility. Finkel et al. [2012] argue that many matching and compatibility claims lack strong empirical support, and that transparent evaluation is essential before making strong claims about algorithm effectiveness. The present work attempts to document and evaluate its scoring procedures transparently, but does not claim that any algorithm will produce satisfying matches in deployment.

TIGERMATCH does not yet implement fairness constraints, differential privacy, or safety reporting mechanisms. These are important requirements for any deployment beyond a research pilot.

7.6. Strategic behavior and incentives

The system relies on participants specifying acceptability sets and importance weights. Even with stable matching theory, participants may have incentives to misreport preferences if they believe it improves their outcomes [Dubins and Freedman, 1981, Roth, 1982]. Participants might broaden acceptability to appear compatible with more people, or set many items to *Mandatory*. Mitigations include:

- limiting extreme weights,
- presenting users with the trade-off between strictness and matchability, and
- designing interfaces that encourage honest reporting.

Future work must address how to present elicitation questions, how to handle strategic behavior, and whether mechanisms should be made robust to manipulation.

8. Conclusion and Future Work

This IW implements a code-grounded pipeline for reciprocal compatibility scoring and cohort-based matching based on structured triad questionnaire data. The implementation includes: (i) a deterministic baseline with item-level acceptability, importance weights, domain multipliers, priority boosts, and finite-overlap penalties (`final_match`); (ii) a probabilistic acceptability model with uncertainty-aware LCB scoring (PAM+LCB); (iii) domain-wise distance-kernel scoring and learning (configurable scoring, soft-gated training, and evolutionary configuration search); and (iv) a merged IDF+LCB pipeline that produces tie-aware preference tiers for matching solvers.

Because IRB approval is pending, this draft reports no human-subject empirical results. Instead, it documents the methodology in a code-linked way and specifies an evaluation framework (Section 6) that can be run verbatim once IRB-approved data collection begins.

8.1. Immediate next steps (within current constraints)

1. **Reproducibility and documentation.** Consolidate evaluation commands, inputs, and outputs into a reproducible script (or Makefile) with pinned environments and fixed random seeds.
2. **Pipeline validation without human data.** Expand unit tests and synthetic-data tests that validate the end-to-end code path (questionnaire parsing $\rightarrow$ scores $\rightarrow$ metrics/matchings), including edge cases such as missing values, ties, and near-zero overlap.
3. **Metric robustness diagnostics.** Add explicit diagnostics for tie handling, missingness, and overlap thresholds to ensure that small preprocessing changes do not produce brittle metric behavior.

8.2. Future work (requires IRB-approved dataset)

1. **Data collection and evaluation.** Recruit a larger cohort (including consenting established couples for offline ground truth) and execute the evaluation framework, including ablations and uncertainty quantification.
2. **Stability-aware learning objectives.** The current learning procedures optimize ranking proxies

(Hit@K, Mutual@K, AUC) rather than stability of the induced matching. A natural extension is to incorporate penalties for predicted blocking pairs under induced matchings.

3. **Ties and incompleteness modeling.** Preference tiers in the merged pipeline (Section 4.5.3) already represent ties. Future work should systematically study stability and optimality under ties/incomplete lists (SMTI).
4. **Fairness, safety, and user experience.** Incorporate fairness constraints (e.g., exposure parity), safety reporting mechanisms, and user-facing explanations that avoid leaking sensitive acceptability constraints.
5. **Evaluation beyond couple recovery.** Offline couple-recovery metrics are only a first sanity check. Ultimately, user-centered outcomes (mutual interest, conversation quality, perceived safety) require surveys, controlled deployments, and qualitative feedback.

Acknowledgements

Thanks to Professor Marcel Dall'Agnol for advising this independent work and for valuable feedback on problem formulation and evaluation methodology. The code accompanying this project was developed with assistance from large language models (LLMs), including OpenAI's Codex, with the GPT-5 series of models.

References

Petrea L. Bach and John M. Malouff. Relationship satisfaction and the big five: Utilizing response surface methodology to evaluate partner similarity. *Personality and Individual Differences*, 233: 112905, 2025. doi: 10.1016/j.paid.2024.112905.

Lester E. Dubins and David A. Freedman. Machiavelli and the gale–shapley algorithm. *The American Mathematical Monthly*, 1981.

Eli J. Finkel, Paul W. Eastwick, Benjamin R. Karney, Harry T. Reis, and Susan Sprecher. Online dating: A critical analysis from the perspective of psychological science. *Psychological Science in the Public Interest*, 13(1):3–66, 2012. doi: 10.1177/1529100612436522.

David Gale and Lloyd S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 1962.

Ruth Gaunt. Couple similarity and marital satisfaction: Are similar spouses happier? *Journal of Personality*, 74(5):1401–1420, 2006. doi: 10.1111/j.1467-6494.2006.00414.x.

Samuel D. Gosling, Peter J. Rentfrow, and William B. Swann. A very brief measure of the big-five personality domains. *Journal of Research in Personality*, 37(6):504–528, 2003. doi: 10.1016/S0092-6566(03)00046-1.

John M. Gottman and Robert W. Levenson. Marital processes predictive of later dissolution: Behavior, physiology, and health. *Journal of Personality and Social Psychology*, 63(2):221–233, 1992. doi: 10.1037/0022-3514.63.2.221.

James A. Hanley and Barbara J. McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982. doi: 10.1148/radiology.143.1.7063747.

Tanya B. Horwitz, Jared V. Balbona, Katie N. Paulich, and Matthew C. Keller. Evidence of correlations between human partners based on systematic reviews and meta-analyses of 22 traits and UK Biobank analysis of 133 traits. *Nature Human Behaviour*, 7:1568–1583, 2023. doi: 10.1038/s41562-023-01672-z.

Robert W. Irving. An efficient algorithm for the stable roommates problem. *Journal of Algorithms*, 1985.

Kazuo Iwama, David Manlove, Shuichi Miyazaki, and Yasufumi Morita. Stable marriage with incomplete lists and ties. In *Proceedings of ICALP*, 1999.

Benjamin R. Karney and Thomas N. Bradbury. The longitudinal course of marital quality and stability: A review of theory, methods, and research. *Psychological Bulletin*, 118(1):3–34, 1995. doi: 10.1037/0033-2909.118.1.3.

Justin A. Lavner, Benjamin R. Karney, and Thomas N. Bradbury. Does couples' communication predict marital satisfaction, or does marital satisfaction predict communication? *Journal of Marriage and Family*, 78(3):680–694, 2016. doi: 10.1111/jomf.12301.

Sointu Leikas, Ville-Juhani Ilmarinen, Markku Verkasalo, and Jan-Erik Lönnqvist. Relationship satisfaction and similarity of personality traits, personal values, and attitudes. *Personality and Individual Differences*, 123:191–198, 2018. doi: 10.1016/j.paid.2017.11.024.

Shanhong Luo and Eva C. Klohnen. Assortative mating and marital quality in newlyweds: A couple-centered approach. *Journal of Personality and Social Psychology*, 88(2):304–326, 2005. doi: 10.1037/0022-3514.88.2.304.

John M. Malouff, Einar B. Thorsteinsson, Nicola S. Schutte, Navjot Bhullar, and Sally E. Rooke. The five-factor model of personality and relationship satisfaction of intimate partners: A meta-analysis. *Journal of Research in Personality*, 44(1):124–127, 2010. doi: 10.1016/j.jrp.2009.09.004.

Henry B. Mann and Donald R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60, 1947. doi: 10.1214/aoms/1177730491.

R. Matthew Montoya, Robert S. Horton, and Jeffrey Kirchner. Is actual similarity necessary for attraction? A meta-analysis of actual and perceived similarity. *Journal of Social and Personal Relationships*, 25(6):889–922, 2008. doi: 10.1177/0265407508096700.

Luis Augusto Pizzato, Thomas Rej, Joel Akehurst, Irena Koprinska, Kalina Yacef, and Judy Kay. RECON: A reciprocal recommender for online dating. In *Proceedings of the ACM Conference on Recommender Systems (RecSys)*, 2010.

Alvin E. Roth. The economics of matching: Stability and incentives. *Mathematics of Operations Research*, 1982.

Uriel G. Rothblum. Characterization of stable matchings as extreme points of a polytope. *Mathematical Programming*, 1992.

Paul Schrodt, Paul L. Witt, and Jenna R. Shimkowski. A meta-analytical review of the demand/withdraw pattern of interaction and its associations with individual, relational, and communicative outcomes. *Communication Monographs*, 81(1):28–58, 2014. doi: 10.1080/03637751.2013.813632.

Shalom H. Schwartz. An overview of the Schwartz theory of basic values. *Online Readings in Psychology and Culture*, 2(1), 2012. doi: 10.9707/2307-0919.1116.

Shalom H. Schwartz, Gila Melech, Arielle Lehmann, Steven Burgess, Mari Harris, and Vicki Owens. Extending the cross-cultural validity of the theory of basic human values with a different method of measurement. *Journal of Cross-Cultural Psychology*, 32(5):519–542, 2001. doi: 10.1177/0022022101032005001.

John H. Vande Vate. Linear programming brings marital bliss. *Operations Research Letters*, 1989.

Rebekka Weidmann, William J. Chopik, Robert A. Ackerman, Marc Allroggen, Emily C. Bianchi, Catherine Brecheen, W. Keith Campbell, Tanja M. Gerlach, Katharina Geukes, Michael P. Grosz, et al. Similarity in personality facets and relationship satisfaction: Testing robustness and identifying trait-specific patterns. *Journal of Research in Personality*, 105:104386, 2023. doi: 10.1016/j.jrp.2023.104386.

Peng Xia, Shuangfei Zhai, Benyuan Liu, Yizhou Sun, and Cindy X. Chen. Design of reciprocal recommendation systems for online dating. *Social Network Analysis and Mining*, 2016.

A. Appendix: Questionnaire summary

This appendix provides a more detailed summary of questionnaire structure. For the authoritative item list and prompts, see `data/questions.json`.

A.1. Representative items

The following examples illustrate how domains are operationalized:

- **Values**: importance of religion, politics, long-term goals, honesty; PVQ-inspired items.
- **Communication**: comfort with conflict, directness, texting frequency, emotional openness.
- **Lifestyle**: sleep schedule, activity level, drinking/smoking, routines.
- **Personality**: TIPI-style statements about extraversion, conscientiousness, emotional stability.
- **Social**: going out versus staying in, group size preferences, spontaneity.
- **Friendship**: integration with friend groups, shared hobbies, time allocation between partner and friends.

B. Appendix: Maximum-weight stable matching formulation

This appendix expands the integer program used for maximum-weight stable matching.

B.1. Inputs: preferences and weights

The solver assumes:

- a set of agents P,
- a strict preference order $\succ_i$ over acceptable partners for each agent i (ties are broken deterministically if needed), and
- a symmetric weight $w_{ij} = w_{ji}$ for each acceptable pair (i, j).

In TIGERMATCH, preferences are derived by sorting candidates by a score $S(i, j)$. Weights are typically set to the same scores (or a monotone transform), so that the objective prefers pairings with higher compatibility.

B.2. Decision variables

For each unordered pair $\{i, j\}$ of agents, define a binary decision variable $x_{ij} \in \{0, 1\}$ indicating whether i is matched to j. We set $x_{ij} = x_{ji}$ conceptually; in implementation it's convenient to index variables by ordered pairs but constrain symmetry or create variables only for $i < j$.

B.3. Matching constraints

Each agent is matched to at most one partner:

$$\sum_{j \neq i} x_{ij} \leq 1 \quad \forall i \in P.$$

(If a perfect matching is desired, one can replace ≤ 1 with $= 1$, but this is inappropriate in many dating settings where being unmatched is preferable to an unacceptable match.)

B.4. Stability constraints

A pair (i, j) blocks a matching if: (i) i is either unmatched or matched to someone they like less than j, and (ii) j is either unmatched or matched to someone they like less than i. To prevent this, we enforce for every acceptable pair (i, j):

$$x_{ij} + \sum_{k \succ_i j} x_{ik} + \sum_{\ell \succ_j i} x_{\ell j} \geq 1.$$

Intuitively, if i is not matched to j and i is not matched to someone preferred over j, then j must be matched to someone preferred over i (or vice versa). Otherwise (i, j) would be a blocking pair.

B.5. Objective

The objective maximizes total weight:

$$\max \sum_{i<j} w_{ij} x_{ij}.$$

When weights are derived from TIGERMATCH compatibility scores, this produces a stable matching that is as compatible as possible under the given preference ordering.

B.6. Practical notes

Two practical considerations matter in small markets:

- **Non-existence and feasibility.** For some preference profiles, stability constraints may make the integer program infeasible. This reflects the fact that a stable matching may not exist (as in stable roommates). In that case, one may relax stability or allow additional structure (e.g., ties or unmatched agents).

- **Sensitivity to tie-breaking.** If many candidates are near-tied in score, deterministic tie-breaking can change which pairs count as blocking. A robust approach is to either keep explicit tie classes or to solve an optimization that accounts for ties directly.

C. Appendix: AUC convention note

The repository function `auc_true_vs_random` computes an AUC-like quantity using a Mann–Whitney formulation, but with ranks assigned in *descending* score order. With that convention, the returned value corresponds to:

$$\mathrm{AUC}_{\mathrm{repo}} = \Pr\left(S_{\mathrm{true}} < S_{\mathrm{rand}}\right) + \tfrac{1}{2}\Pr\left(S_{\mathrm{true}} = S_{\mathrm{rand}}\right),$$

which is the complement of the standard AUC:

$$\mathrm{AUC}_{\mathrm{std}} = 1 - \mathrm{AUC}_{\mathrm{repo}}.$$

To prevent confusion, the evaluation framework in Section 6.2 defines and reports $\mathrm{AUC}_{\mathrm{std}}$ explicitly.

For completeness, the relevant code paths are:

- `algorithm/eval/evaluate_couples.py::auc_true_vs_random`,
- `algorithm/eval/harness.py::auc_true_vs_random`,
- and a duplicate implementation in `algorithm/paper/compare_all.py::auc_true_vs_random`.

D. Appendix: Reproducibility

The following commands rebuild key artifacts (from the `algorithm/` directory) once IRB-approved private inputs `../export.csv` and `../couples.csv` are available locally:

- Build question metadata:

```
python3 tools/build_meta_from_repo.py \
  --questions_json ../data/questions.json \
  --export_csv ../export.csv \
  --out meta.json
```

- Evaluate baseline:

```
python3 eval/evaluate_couples.py \
  --export_csv ../export.csv \
  --couples_csv ../couples.csv \
  --meta_json meta.json \
  --questions_json ../data/questions.json \
  --outdir out
```

- Evaluate PAM variants:

```
python3 scoring/score_pairs_cli.py \
  --export_csv ../export.csv \
  --meta_json meta.json \
  --questions_json ../data/questions.json \
  --delta 0.10 \
  --out scores.csv

python3 eval/evaluate_pam.py \
  --export_csv ../export.csv \
  --couples_csv ../couples.csv \
  --meta_json meta.json \
  --questions_json ../data/questions.json \
  --delta 0.10 \
  --out out/summary_pam.csv
```

- Build distance-kernel feature tensors:

```
python3 preprocess/normalize.py \
  --export_csv ../export.csv \
  --meta_json meta.json \
  --out norm.json

python3 features/build_pair_tensors.py lcbs \
  --export_csv ../export.csv \
  --meta_json meta.json \
  --questions_json ../data/questions.json \
  --delta 0.10 \
  --out lcbs.npz

python3 features/build_pair_tensors.py dists \
  --export_csv ../export.csv \
  --meta_json meta.json \
  --norm_json norm.json \
  --out dists.npz

python3 features/make_features.py \
  --lcbs lcbs.npz \
  --dists dists.npz \
  --out features.npz
```

- Train the soft-gated model:

```
python3 ml/soft_gated_trainer.py \
  --features features.npz \
  --couples_csv ../couples.csv \
  --neg_k 20 \
  --epochs 50 \
  --outdir out_soft

python3 eval/harness.py \
  --features features.npz \
  --couples_csv ../couples.csv \
  --config out_soft/best_config.json \
  --outdir out_soft \
  --topk 3
```

- Run evolutionary search:

```
python3 ml/evolutionary_trainer.py \
  --features features.npz \
  --couples_csv ../couples.csv \
  --pop 80 \
  --gens 120 \
  --topk 3 \
  --outdir out_evo

python3 eval/harness.py \
  --features features.npz \
  --couples_csv ../couples.csv \
  --config out_evo/best_config.json \
  --outdir out_evo \
  --topk 3
```

- Run merged IDF+LCB pipeline and matching:

```
python3 matching/merged_pipeline.py \
  --export_csv ../export.csv \
  --couples_csv ../couples.csv \
  --meta_json meta.json \
  --outdir out_merged \
  --rho 0.35 --delta 0.10 --tau 0.05 --mu 0.03 --topk 2
```