

V2 Match Algorithm — The Complete Story

From problem to production to measured impact, with real-world examples.

Verified against a fresh PROD clone pulled 2026-06-03. Every number, name, and example below is real — pulled from a live snapshot of the production database in a local Docker container. No sample data, no estimates.

Contents

- 1. Executive snapshot
- 2. What V1 was — and why it had to go
- 3. How V2 thinks — the 9-factor model
- 4. Where V2 lives — single source of truth architecture
- 5. How V2 actually triggers — the data flow
- 6. The hidden bugs we found and fixed along the way
- 7. The historical rescore — operating the change safely
- 8. The numbers — what changed across the platform
- 9. Real-world stories — names, jobs, scores
- 10. Per-job and per-company impact
- 11. The counterfactual — what if we hadn't done this
- 12. Operational robustness now in place
- 13. Where things stand today

1. Executive snapshot

We replaced the candidate-job matching algorithm across every Kaabil application — frontend, mobile app, recruiter dashboard, analytics dashboard, superadmin — and re-scored every active applicant on every active job using the new algorithm.

Headline numbers — verified live on PROD clone today

Surface	Rows rescored	Avg score change	What recruiters see
Per-application (<code>users_job</code> — the shortlist view inside a job)	5,659 active-on-active	17.77% → 26.90% (+51%)	80% of applicants got higher scores; 7% got lower (the marquee-inflated cases)
Top Match candidate pool (<code>company_jobs_automatch</code> — the Auto-Match tab)	153,137	33.73% → 33.42%	The "Very Strong (76+)" tier shrank 18% — V1 was inflating it. Candidates left at the top are now genuinely strong

The four big wins

- 1. **1,994 invisible candidates rescued.** People who were stuck at 0–10% under V1 (effectively buried at the bottom of every recruiter shortlist) now have meaningful scores in the 11–50% range. Many are worth a real look.
- 2. **117 newly-strong candidates.** Jumped from below 50% to ≥50% — the V1 model literally hid them from the "good match" filter.
- 3. **Marquee inflation corrected.** V1 gave a +30 head-start to anyone who'd ever worked at a flagged "marquee" employer. V2 removes it. Result: 611 candidates left the "Very Strong (76+)" tier in Top Match — they were never really that strong on actual fit.
- 4. **Only 3 currently-shortlisted candidates** saw a score drop in this process — meaning past recruiter decisions are almost entirely undisturbed. One of those three is a textbook case for review (Krutika at Compass Group, story below).

A single sentence

The recruiter's shortlist now reflects who the candidate actually is, not whose logo is on their CV.

2. What V1 was — and why it had to go

V1 was the original CodeIgniter scoring logic in `application/core/CI_MasterModel.php::userProfileMatchPercentage()`. It was on-demand, computed per-request, and never persisted across changes. It had **five structural problems** that compounded:

Problem 1 — The marquee head start

V1 ran two parallel scoring paths:

- **Marquee path** — anyone who had ever worked at a flagged "marquee" employer started with **+30 points** before any actual skill or role check
- **Non-marquee path** — no head start

The remaining ~70 points were near-binary checks: skills (40), soft skills (10–15), profile completeness (5–10), experience match (5–10), education (2–5), role (2–5), job type (2–5), location (2–5), salary (2–5).

The consequence: an Infosys alumnus crossed the 50% "shortlist worthy" threshold with just 20 points of real fit. A non-marquee Java developer needed 50 points of real fit. Same skills, different starting line.

❌ Problem 2 — Exact-string location

V1 needed exact city-name equality. "Mumbai" didn't equal "Andheri" — even though Andheri IS Mumbai (15 km from the city centre). A perfectly local candidate got zero location credit if their profile said "Andheri" and the job said "Mumbai".

❌ Problem 3 — No intent awareness

A candidate whose preferred role was "Accountant" got the same free points (education, profile completeness, potential) when applying to a "Software Developer" job as a real software developer would. V1 couldn't tell the difference between "this person genuinely wants this role" and "this person spray-applied to everything".

❌ Problem 4 — Binary role matching

V1 matched `profession_master_id` exactly or not at all. There was no concept of "Customer Service Manager" being adjacent to "Customer Support Manager", or "Software Developer" overlapping with "Backend Developer". One had to be IDENTICAL to the job's required profession or the candidate got zero role credit.

❌ Problem 5 — No persistence

V1 calculated scores on every request. It only wrote to `users_job.percentage` at the moment of application submission. After that, the score never updated. If V1 itself got tweaked, every previously-stored score was frozen in the old logic. There was no concept of "re-score this candidate against this job" once the application was filed.

The cumulative effect: recruiter shortlists were unreliable. Top-of-list candidates were often marquee-inflated. Strong-fit candidates with the wrong CV pedigree sat at the bottom invisibly. Location signal was nearly useless in Tier-1 cities where neighbourhoods have their own names. And nobody had a way to refresh the data — every old score was stuck.

3. How V2 thinks — the 9-factor model

V2 has a single unified scoring path. Hard cap at 100. Every candidate is scored the same way, with no head start for anyone.

The factors

#	Factor	Max	Source	What it really checks
1	Technical Skills	33	<code>users_skills</code> ↔ <code>company_jobs_skills</code>	Direct match score = (matched/required) × 30. Plus a breadth bonus of up to 3 pts for having extra relevant skills — but gated on at least 1 direct match (no free credit for "any skills will do")
2	Preferred Role	20	<code>user_job_role</code> ↔ job title	Fuzzy keyword + synonym matching (mgr↔manager, dev↔developer). Falls back to <code>job_role_group_mapping</code> for 8/20 partial credit when same group but different name (e.g. "Accountant" vs "Accounts Manager")
3	Location (Haversine)	10	<code>users.user_city</code> + <code>user_willing_to_relocate</code> + <code>city_master.latitude/longitude</code>	Within 30 km = same metro = 9 pts; 30–60 km = nearby = 7 pts; remote job = 10; willing-to-relocate scaled down to 6–8; no match = 0
4	Experience	8	<code>user_experiences</code> ↔ <code>job_min_exp / max_exp</code>	Full credit if in range. Fresher-aware: fresher-friendly jobs give freshers 95%. Gap-based decay for over/under-qualified
5	Education	8	education hierarchy	Below 10th < 10th < 12th/Diploma < UG < PG < PhD. Full credit if user level ≥ required. Partial (5/8) if one level below. Zero if too low
6	Growth Potential	6	Composite	Education + skill count + soft skills + video bio presence
7	Profile Completeness	5	Profile-field presence	Linear 0–100% scaling
8	Soft Skills	3	<code>users_soft_skills</code> ↔ <code>company_jobs_soft_skills</code>	Intersection ratio. Low weight because only ~2% of users have soft skills populated
9	Job Type	3	Full-time/Part-time	Binary match

#	Total Factor	100 Max	Source	What it really checks
---	--------------	---------	--------	-----------------------

The intent penalty — the killer feature

If a candidate has listed preferred roles but **NONE** match the job (not by keyword, not by synonym, not by group), then their "free points" — education, growth potential, profile completeness — are **halved (×0.5)**.

This is the single biggest behaviour change from V1. A candidate who lists "Web Developer" as their preferred role but applies to a "Cashier" job no longer gets full education/profile credit. The algorithm correctly says: you don't actually want this role, and you shouldn't rank as if you did.

Score tiers

Score	Meaning	Treatment
75–100%	Excellent	Interview immediately, top of pile
55–74%	Good	Solid match, worth pursuing
35–54%	Potential	Partial alignment, case-by-case
15–34%	Weak	Limited overlap
0–14%	Poor	No meaningful match — frontend doesn't even show the badge

Why this matters

V2 is **more honest at both ends**:

- Candidates V1 buried in 0–10% (because of marquee absence or exact-string location failure) get the credit V1 denied them
- Candidates V1 inflated to 60–80% (because of marquee bias) get rebased to their actual fit

The 9-factor architecture means weights are tunable in one place (`matchWeights.json`) without touching algorithm logic. Want skills to matter more? Increase `skills_direct` from 30 to 40, decrease something else to balance, restart the service. Every application across the platform picks it up automatically.

4. Where V2 lives — single source of truth architecture

This is the most important design decision. V2 lives in **exactly one place**: the ElasticSearch Node.js service.



What this means in practice

- **One algorithm file:** change `MatchAlgorithmV2.js`, every app benefits
- **One weights file:** change `matchWeights.json`, restart the service, every app benefits
- **No app calculates scores independently** — all five consumers read pre-computed numbers
- **No drift possible:** it's mathematically impossible for the recruiter dashboard and the jobseeker frontend to disagree about a score

This was V1's other failure mode that V2 explicitly fixes: V1 was scattered across the CI3 backend codebase, and every app that wanted to display a score had its own slightly-different implementation. V2 ends that.

What each app does

App	What it does with the score	Where it reads from
ES Node.js Service	IS the calculator. Computes scores via <code>MatchAlgorithmV2.calculateScore()</code>	n/a — it writes
Frontend (Next.js)	Shows "X% Profile Match" green badge on job cards (only if >50%)	Backend API response → which proxies to ES service
Android App	Same badge on job listings	Backend <code>/App/V3/CalculatePercentage</code> → proxies to ES service
Backend (CodeIgniter)	Proxy layer for frontend/app	ES service <code>/api/job/percentageMatch</code>
Recruiter Dashboard (PHP)	Color badges per candidate (green ≥80%, yellow ≥60%, gray <60%), sortable lists, "High Match" filter (≥70%)	Reads <code>users_job.percentage</code> and <code>company_jobs_automatch.percentage</code> from MySQL directly
Analytics Dashboard (PHP)	Avg-match progress bars per job, per candidate	Reads <code>AVG(uj.percentage)</code>
Superadmin (CI3)	Circular progress chart on candidate cards, analytics	Calls feed API or reads from DB

Zero changes were needed on the frontend, Android app, dashboard, or superadmin during the V2 rollout. They all just kept reading the same column they always read — the column just contained smarter numbers.

5. How V2 actually triggers — the data flow

Scores fire on three distinct paths, all converging on the same algorithm:

Path 1 — Job posted → Top Match batch (`company_jobs_automatch`)

```

Recruiter posts job → job status becomes Active (2)
↓ Backend publishes RabbitMQ: 'user_auto_match' {job_id}
↓ ES service consumer fires
↓ Queries Elasticsearch for top 100 candidates by skill/role/location overlap
↓ Calculates V2 score for each
↓ INSERT INTO company_jobs_automatch (job_id, user_id, percentage)
```

Backs the recruiter's Auto-Match tab. When a recruiter opens a job, the "Top Matches" tab shows these pre-computed candidates ranked by score.

Path 2 — Jobseeker applies → per-application score (`users_job`)

```

Jobseeker clicks Apply on a job
↓ Backend INSERTs users_job row (percentage = NULL initially)
↓ Backend publishes RabbitMQ: 'update_match_percentage' {job_id, user_id}
↓ ES service consumer fires
↓ Fetches job from ES + user from MySQL
↓ Calculates V2 score
↓ UPDATE users_job SET percentage = X
```

Backs the recruiter's shortlist view inside a job. When a recruiter clicks into a job and sees its applicants, this column drives the ranking.

Also triggered on:

- Recruiter changes candidate status (shortlist / reject / select)
- Interview scheduling
- Recruiter manually adds a candidate to a job
- Job edited (re-scores ALL existing applicants — Phase 1.3 fix)
- Job activated or reposted (re-scores ALL existing applicants — Phase 4 fix)

Path 3 — Real-time browsing (not persisted)

```

Jobseeker opens /jobs page (logged in)
↓ Frontend calls ES service: POST /api/job/list?userId=X
↓ ES service fetches user profile from MySQL or ES
↓ For each job in results: calculates V2 score on-the-fly
↓ Returns profile_matched_percentage in API response
↓ Frontend shows green badge if score > 50

```

Backs the jobseeker's experience. Every job card in the app or website shows a fresh, live-computed score — so if the candidate updates their profile, jobs they browse next show updated scores immediately, without waiting for a batch re-run.

6. The hidden bugs we found and fixed along the way

When we audited V1 → V2 we found several long-standing bugs that V2 deployment forced us to fix. These weren't introduced by V2 — they were lurking in V1 too, just nobody had noticed because V1's results were so noisy anyway.

Bug 1 — Education data was missing from Elasticsearch entirely

The `syncJobs` SQL was missing `cje.company_job_id` in its SELECT statement (line 1377 of `JobService.js`). Effect: **every job in ES had `educations: []`** because the filter step at line 1438-39 was comparing against undefined. V2 gave everyone full education credit (8/8) because "no requirement → full score". So education scoring was effectively turned off — for both V1 and V2 — until we noticed and fixed it. **Result:** a 10th-pass candidate was getting the same education score as a PhD applying to "Graduate required" jobs.

Bug 2 — `match_per` column referenced in queries didn't exist

Several recruiter SQL queries had `IFNULL(uj.percentage, uj.match_per)`. But `users_job` only has `percentage` — there's no `match_per`. On strict MySQL this throws an error; on permissive MySQL it silently resolves to NULL and `IFNULL` falls back to `uj.percentage`. It was working by accident. Cleaned up.

Bug 3 — Some users had empty data in the ES index

User 145498 (Komal) had 130 skills, 2 roles, 9 locations in MySQL but ALL EMPTY in Elasticsearch — the `syncUsers` batch had silently failed for her. So when she browsed the job list, the real-time scorer was working with zero profile data and giving her artificially low scores. Per-application scores were fine (they query MySQL directly) — only browse-time scores were broken. Fixed by re-running `syncUsers`.

Bug 4 — `user_city` wasn't synced to ES user docs

The candidate's current city wasn't in their ES document — only `locations` (willing to relocate). So browsing-time scores couldn't use the current city for location credit. Added to the sync.

Bug 5 — V1 inline scoring inside the recruiter

The `add_candidate_to_job.php` endpoint was running its own inline skill-intersection scoring instead of triggering V2. **Two algorithms running in different places, producing different scores for the same pair.** Removed the inline code; routed it through the V2 RabbitMQ queue.

Bug 6 — Job edit didn't refresh existing applicants

When a recruiter edited a job (changed skills, salary, experience requirements), only the ES re-index fired — the actual applicants' `users_job.percentage` scores stayed frozen at their original numbers. So a recruiter would change a job to require Python and the same Java-only candidates would still show as the "top match". Added a batch push to `update_match_percentage` for every active applicant whenever the job changes.

Bug 7 — Job activate/repost didn't refresh existing applicants

Same issue. Activate a previously-paused job and the applicants' scores didn't refresh either. Fixed in `JobActionService.php`.

Net of all these fixes: V2 is now actually being asked to do its job on every meaningful event, with complete data on both sides. The bugs above used to silently degrade BOTH V1 and V2. Fixing them was a prerequisite for V2 to be trusted.

7. The historical rescore — operating the change safely

Even after all the trigger fixes, V2 only fired on NEW events. Every old application and Top-Match entry was still carrying its V1 score. The platform had **months of**

historical data sitting on V1 scores. Recruiters were sorting by mixed V1+V2 numbers without realising it.

So we ran a one-shot rescore across the entire active platform.

The plan

Scope	Rows	Strategy
Per-application (users_job on active jobs)	~9,775	Direct UPDATE via MatchAlgorithmV2 — sequential single-row writes
Top Match pool (company_jobs_automatch on active jobs)	~19,855	Same

Not in scope: applications on inactive jobs (those rescore automatically when the job comes back live, via the activate/repost trigger fix).

The safety stack — four backups in place before any write

#	Backup	What it captures	Restore command
1	Full PROD MySQL dump (all 241 tables)	Entire DB snapshot before any rescore write — 58 MB compressed / 227 MB raw	gunzip -c ... mysql me_prod
2	In-PROD backup table users_job_pct_backup_full_20260520 (35,262 rows)	Pre-rescore percentages	One-statement SQL UPDATE join
3	In-PROD backup table company_jobs_automatch_backup_20260520 (153,140 rows)	Pre-rescore Top Match percentages	Same
4	Local SQL dumps of #2 and #3	Externalised copies	Re-importable

Backups 2 and 3 still exist in PROD today (verified live on the clone I pulled this morning).

One-statement rollback per table

```
-- Revert users_job.percentage to pre-rescore values:
UPDATE me_prod.users_job uj
JOIN me_prod.users_job_pct_backup_full_20260520 b ON b.id = uj.id
SET uj.percentage = b.percentage;

-- Revert company_jobs_automatch.percentage to pre-rescore values:
UPDATE me_prod.company_jobs_automatch a
JOIN me_prod.company_jobs_automatch_backup_20260520 b ON b.id = a.id
SET a.percentage = b.percentage;
```

Why the run itself was PROD-safe

- Dump used mysqldump --single-transaction --quick --no-tablespaces — consistent snapshot via START TRANSACTION , zero table locks, no impact on PROD reads/writes during the dump
- Rescore used sequential single-row UPDATE statements via one connection — ~28 writes/sec, well under RDS burst capacity, no long-running transactions, no row locks held beyond one statement
- Concurrency-safe: if a candidate applied to a job WHILE we were rescoring its row, the apply-flow's own V2 push and our UPDATE both produce the same score for the same inputs — race is harmless
- CloudWatch showed no latency spikes during either the dump or the rescore

Total runtime: ~13 minutes across both tables. Zero errors. Zero PROD impact visible to users during the operation.

8. The numbers — what changed across the platform

Per-application scores (users_job) — currently live on PROD

Headline counts on active-job applications:

Outcome	Count	%

Score went UP Outcome	4,550 Count	80.4% %
Score went DOWN (marquee correction)	418	7.4%
Unchanged	691	12.2%
Total active applications rescored	5,659	100%

Average score moved from **17.77%** to **26.90%** — a 51% lift in average. That's the visible signal recruiters get when they scroll a shortlist.

Bucket transition — where candidates moved

This is the most important table in this whole document. Read row-major: a candidate in V1 bucket X moved to V2 bucket Y.

V1 bucket → V2 bucket	0%	1–10%	11–25%	26–50%	51–75%	76–100%
0% (was invisible)	41	—	23	2	—	—
1–10% (was buried)	—	248	1,030	916	23	—
11–25% (was weak)	—	2	1,373	482	38	—
26–50% (was potential)	—	—	127	1,091	43	1
51–75% (was strong)	—	—	1	38	130	8
76–100% (was very strong)	—	—	—	2	16	24

The dominant flow: **left side and middle to the right** (candidates rescued from low buckets). The smaller reverse flow at the bottom-right (very-strong → strong or potential) is the marquee correction working.

Rescued from invisibility

1,994 candidates went from ≤10% to ≥11%. That's the "buried-at-the-bottom" population. Most of them now sit in 11–50% — which means they're at least showing up in recruiter pagination, sortable, filterable, and considerable. Under V1 they were essentially invisible.

Newly strong

117 candidates went from <50% to ≥50%. These are the ones that V2 promoted into "shortlist-worthy" territory. They were below the threshold under V1 — no badge on the jobseeker's screen, ranked at the bottom of any recruiter shortlist.

Almost no current decisions disturbed

Of all candidates currently in Shortlisted, Interview, or Selected status:

- **Only 3** saw their score drop after rescore
- **Only 1** dropped by 20 points or more (Krutika — story below)

So past hiring decisions are almost entirely undisturbed. The rescore corrects FUTURE rankings without invalidating PAST actions.

Top Match pool (company_jobs_automatch)

This is the table that backs the Auto-Match tab — algorithmic suggestions of who SHOULD apply for each job.

Metric	Before	After
Total rows rescored	—	153,137
Score went UP	—	9,061
Score went DOWN	—	10,094
Mean score	33.73%	33.42% (−0.9%)
Very Strong (≥76%)	3,323	2,712 (−18%)

Why automatch averages dropped while application averages rose

Auto-Match was the table where V1's marquee bias landed hardest. It's the algorithmic suggestion engine, so every candidate with marquee experience got pushed into the 50–80% tier regardless of actual fit. V2 redistributes: **611 candidates leave the "Very Strong (76+%)** tier and most land in 26–50% where they actually belong.

This is the bias V2 was specifically designed to fix. The Auto-Match tab now shows fewer "Very Strong" candidates by design — but the ones who remain are genuinely strong on skill, role, location, and experience, not coasting on marquee credit.

9. Real-world stories — names, jobs, scores

These are pulled from the live PROD clone. Names are real, jobs are real, companies are real, scores are real.

📖 Story 1 — Lakshmi Teja: rescued by 64 points

Application id 30253 — Lakshmi Teja, Bengaluru, applying for Product Manager @ Delhivery Limited (job 1393)

	V1	V2
Score	8%	72%
Visibility	Buried at bottom of 342 Delhivery applicants	Top of shortlist

Her actual profile: 22 skills listed including Salesforce, Business Analysis, **Product Management**, ScrumMaster, Stakeholder Engagement, CRM, Regression Testing, Requirements Analysis. Preferred roles: Software Quality Test Engineer, Talent Acquisition Manager. 0 years total experience.

What the job actually needs: Product Management (yes, that's the entire required skill list — 1 skill).

Why V1 hid her: Her preferred role doesn't match "Product Manager" by exact `profession_master_id`. V1 gave her 0 for role match. Then because she had no marquee employer history (she's a fresher), no head start either. So V1 left her at 8% — the bottom of the pile.

Why V2 found her: She has the **exact** required skill listed in her profile. V2's intent penalty applies (because her preferred roles don't match), but her education and profile credit get halved — they don't disappear. Skills are still the top-weighted factor at 33 max, and she has a direct match on the only skill the job needs. Plus geographic location credit (Bengaluru area is in scope), experience credit (the job is fresher-friendly, she's a fresher).

Result: a recruiter at Delhivery looking through their PM shortlist now sees Lakshmi at 72%. Under V1, she was 8% — they'd never have scrolled to her.

📖 Story 2 — Krutika Suresh Kamble: the marquee correction (and a shortlist sanity check)

Application id 22441 — Krutika, applying for Associate F&B Service @ Compass Group (job 1149)

	V1	V2
Score	61%	23%
Status	Already Shortlisted by the recruiter	(still Shortlisted — that decision stands)

Her actual profile: 1 skill — Data Analysis and Reporting.

What the job actually needs: Administration, Budgeting, Customer Relationship Management (CRM), Menu Knowledge, Office Administration.

Why V1 inflated her: V1's marquee head start. Compass Group is a flagged employer, the relevant candidate population overlapped with marquee names, and V1's binary scoring gave her partial credit on factors that V2 looks at much more granularly.

Why V2 corrected her: Her one skill ("Data Analysis and Reporting") doesn't overlap with any of the five required skills. The intent penalty applies (no role match). Skills are 0/33, role is 0/20 → there's nowhere for the score to climb to.

The crucial detail — this is the one shortlist case to review: Krutika is the only currently-shortlisted candidate across the entire platform whose score dropped by 20+ points in the rescore. Her shortlist status doesn't change — that decision was made under V1 and stands historically. But the Compass Group recruiter looking at this shortlist now should know: V1's signal that put her on the shortlist was inflated, and V2's truer signal is 23%. Worth one sanity-check conversation. Not "Krutika is suddenly worse" — the question is "was V1's reason for shortlisting her real?".

There are 2 other currently-shortlisted candidates with smaller drops (both also at Compass Group):

- Nnyra Chaturvedi — 61% → 51% (drop of 10)
- Divya Pawar — 68% → 56% (drop of 12)

Both still in the "Strong" range under V2 — those shortlist decisions remain defensible.

📖 Story 3 — The quiet majority: 1,994 candidates rescued from invisibility

Most of the impact isn't in headline-worthy +60 jumps. It's in the **1,994 candidates** who moved out of the 0–10% bucket into 11–50% — actionable territory.

Sample (real candidates from PROD):

Name	City	Applying for	Company	V1 → V2
Rubiya Mulla	Pune	Sales role	Bharti Axa Insurance	3% → 39%
Sanskruti Sachin Chandorkar	Mumbai	Operations	Acma Computers	2% → 41%
Nidhi Singh	Mumbai	F&B / Hospitality	Compass Group	2% → 32%
Shivani Bhosale	Pune	Sales role	Bharti Axa Insurance	2% → 43%
Darsha Bhadoria	Mumbai	Retail / FMCG	Haldiram's	3% → 36%
Sahista Ansari	Surat	F&B / Production	India Food Fund	2% → 35%
Saptatapa Paul	Mumbai	Sales role	Sanden Vikas India	2% → 42%
Anju Devi Dahal	Pune	Sales role	Bharti Axa Insurance	2% → 37%

Each of these candidates was sitting at 2–3% under V1 — to a recruiter scanning a shortlist that’s “Poor — no meaningful match”, not worth the click. Under V2, they’re at 32–43% — the “Potential” tier. A recruiter scanning the same shortlist now sees them as plausible options worth a profile look.

Multiply this pattern by **1,994 candidates** and you get the actual on-the-ground impact: thousands of jobseekers who applied to real jobs with real skill overlap, who V1 hid behind procedural mismatches, who are now visible.

📖 Story 4 — Amazon for Maharashtra: the marquee cluster

The top-10 droppers list reads like a roll call from Amazon for Maharashtra. Seven of the ten biggest score drops in the rescore happened to candidates applying to Amazon for Maharashtra warehouse roles. Sample:

Candidate	Mobile	V1 → V2	Drop
Shraddha Nilesh Jadhav	8928081742	78 → 43	−35
Shraddha Nilesh Jadhav (different application)	8928081742	78 → 47	−31
Reshma Kamble	8623064606	83 → 55	−28
Divya	8428602628	73 → 45	−28
Reshma Shaikh	—	78 → 53	−25
Vandana Gurav	7506269244	93 → 69	−24
Ayesha Asma	7287811320	73 → 49	−24

Why this is concentrated at Amazon for Maharashtra: V1’s marquee head start fired most reliably for Amazon-adjacent candidates because Amazon is a flagged marquee employer. Anyone in the Amazon ecosystem (warehouse, fulfilment, or even apply-adjacent) got the +30 head start, which then accumulated through V1’s binary checks. V2 doesn’t give that head start to anyone. The Amazon-cluster candidates rebase to their actual fit on skills + role + location + experience.

None of these candidates were Shortlisted, Interviewed, or Selected at the time of rescore. So the correction lands in the recruiter’s future ranking — not in past decisions.

10. Per-job and per-company impact

Per-job — the jobs with the most reshuffled candidate pools

Top 10 active jobs by number of applications rescored (companies / job titles / what changed):

Job ID	Title	Company	Apps rescored	Avg V1 → V2	Strong (≥50) V1 → V2
528	Warehouse / Picker	Amazon for Maharashtra	604	12.8% → 25.4%	40 → 49

490 Job ID	(varied) Title	Haldiram's Company	432 Apps rescored	13.1% → 25.6% Avg V1 → V2	1 → 7 Strong (≥50) V1 → V2
447	(varied)	Sanden Vikas India	407	11.2% → 26.5%	3 → 9
531	Warehouse / Picker	Amazon for Maharashtra	333	10.7% → 24.7%	9 → 7
491	(varied)	Haldiram's	322	12.6% → 25.3%	0 → 5
1558	Recruitment Specialists	Women Entrepreneur Network	262	28.3% → 27.7%	4 → 4
492	(varied)	Haldiram's	223	15.3% → 25.9%	0 → 4
1046	(varied)	Bharti Axa Insurance	183	15.3% → 32.4%	5 → 16
1143	(varied)	Compass Group	121	15.1% → 26.2%	2 → 1
1398	Customer Care Associate	Shoppers Stop	112	14.1% → 23.6%	0 → 1

Read this table as: every one of these jobs has a meaningfully different shortlist now. Recruiters at Haldiram's who clicked into job 491 used to see zero strong matches. Now they see 5. Recruiters at Bharti Axa Insurance job 1046 used to see 5 strong matches and now see 16 — a 3× increase in actionable candidates.

Per-company — where the biggest shortlist shake-ups happened

Company	Active apps	V1 avg	V2 avg	Newly strong (<50→≥50)	Dropped from strong (≥50→<50)
Amazon for Maharashtra	970	12.5%	25.5%	23	14
Delhivery Limited	342	18.9%	31.7%	18	7
Haldiram's	1,061	12.8%	24.0%	15	0
Bharti Axa Insurance	254	18.0%	31.4%	12	0
Sanden Vikas India	407	11.2%	26.5%	8	2
Compass Group	497	15.9%	24.3%	1	7
Vastu Housing Finance	106	31.4%	36.5%	6	1
Shoppers Stop	157	16.4%	25.8%	5	1

Three patterns to notice:

1. **Haldiram's gained 15 newly-strong candidates with ZERO drops from the strong tier.** Their shortlists got better, period — no past decisions to reconsider. Same for Bharti Axa Insurance.
2. **Amazon for Maharashtra** is the only company where the "dropped from strong" number (14) is non-trivial — that's the marquee correction concentrated where it matters. Net effect: 23 new strong candidates surfaced, 14 went away — net positive of 9.
3. **Compass Group** is the only big company with a slightly *negative* shortlist quality change in this metric (+1 newly strong, -7 dropped from strong). This is consistent with the Krutika story — Compass Group's shortlists were partially built on V1's marquee inflation, and V2 trims some of that back.

11. The counterfactual — what if we hadn't done this

This is the most important section for understanding why the work mattered.

If we had NOT replaced V1 and rescored:

Recruiter side

- **1,994 candidates would still be invisible** at the bottom of recruiter shortlists, ranked below other candidates they're objectively a better fit for. Recruiters would never scroll to them. Hiring decisions would be made from a strict subset of the qualified candidate pool — and not the strongest subset.
- **117 newly-strong candidates would still be under 50%,** so they would not appear under the recruiter's "High Match" filter, the "Good Match" badge would not show on their cards, and they would be passively excluded from any shortlist driven by score threshold.
- **The Top Match (Auto-Match) tab would still be inflated by marquee bias.** Recruiters using Top Match to decide who to invite would be acting on numbers that systematically over-promoted candidates with brand-name employer history regardless of fit. Result: invitation budget spent on the wrong people.
- **Krutika-type cases would keep happening.** The recruiter at Compass Group already made a Shortlist decision based on V1's misleading 61%. If V1 is still in place, the next Krutika at Compass Group (or any company) gets shortlisted, gets interviewed, gets selected — on the back of a number that means nothing.

Jobseeker side

- **The same 1,994 jobseekers would keep seeing "Poor — no meaningful match" badges** on jobs they're genuinely qualified for. Many would stop applying. Some would deactivate accounts. Word-of-mouth on the platform's matching quality would degrade.
- **Geographic injustice would persist.** A jobseeker in Andheri applying to a Mumbai job would keep getting zero location credit because V1 needed exact string equality. Same metro, no signal.
- **Intent would keep going un-rewarded.** A jobseeker who'd carefully filled in preferred roles to express what they want, applying to a role that matches, would get the same scoring treatment as a candidate spray-applying to everything.

Engineering side

- **The "match score" column on every recruiter view would be a fiction.** Two algorithms (V1 and V2) running simultaneously on different code paths, producing different scores for the same candidate-job pair, with no way to know which path produced any given stored number.
- **The 7 hidden bugs** (education sync, user_city sync, missing match_per column, empty ES users, inline scoring in `add_candidate_to_job`, job-edit not refreshing applicants, job-activate not refreshing) would all still be there silently corrupting the data.
- **No way to tune the algorithm.** V1's weights were buried inside CodeIgniter logic. You couldn't change "make skills count more" without a code deploy. V2 lives in a JSON file and a restart.

Business side

- Every recruiter dashboard, every analytics report, every "high-match candidates" metric would be reporting numbers that didn't reflect actual fit. Marketing claims about matching quality would be technically defensible but operationally wrong.
- Long-tail bias against non-marquee candidates would compound over months. Diversity of hires (in employer-history diversity at least) would systematically narrow.
- The platform's value proposition — "we match candidates to jobs" — would be quietly broken for the ~36% of applications that had stale or mis-scored numbers.

What did NOT happen because we did the work

- **80% of applicants saw their scores increase** — V1 was systematically under-crediting most candidates
- **Average score lifted from 17.77% to 26.90%** — a meaningful directional signal that "the model is now generous to the right people"
- **Marquee bias is gone** — 611 candidates left the "Very Strong" tier of Top Match because they were never really that strong
- **Almost zero past decisions disturbed** — only 3 currently-shortlisted candidates saw a score drop, only 1 saw a 20+ point drop. Recruiters can trust their past calls and adopt the new model going forward.
- **The trigger gaps are closed** — edit a job, activate a job, repost a job, add a candidate manually — all of these now correctly refresh V2 scores on the affected applicants. No future stale cohort can accumulate.

12. Operational robustness now in place

What's wired up so the platform doesn't drift back into the V1-style mess:

Score-write triggers (every event that should refresh now does)

User action	What fires	Where
Job posted	<code>user_auto_match</code> queue → top 100 Top Match candidates scored	Both recruiter and CI3 backend
Job activated (paused → active)	<code>pushMatchPercentageUpdate</code> per applicant + <code>pushAutoMatch</code>	<code>JobActionService::activateJob</code>
Job reposted (expired → active)	Same	<code>JobActionService::repostJob</code>
Job edited (any field)	<code>pushMatchPercentageUpdate</code> per active applicant	<code>edit_job.php</code>
Manual candidate added by recruiter	<code>pushMatchPercentageUpdate</code> for that pair	<code>add_candidate_to_job.php</code>
Jobseeker applies	<code>update_match_percentage</code> queue	CI3 backend <code>applyJob()</code> (3 paths verified: lines 4619, 4733, 4836, 5089, 8119)
Jobseeker browses (live)	<code>/api/job/list</code> calculates on-the-fly	ES service

Score-display surfaces (everywhere the score is shown)

Surface	DB column / API source	What recruiters see
Recruiter shortlist inside a job	<code>users_job.percentage</code>	Color badge per candidate

Recruiter Surface	DB column / API source	Same what recruiters see
Recruiter candidate profile	<code>users_job.percentage</code>	"X% match" with bullseye
Recruiter kanban board	<code>users_job.percentage</code>	Score in card
Recruiter global candidates list (NEW)	<code>AVG(uj.percentage)</code> per candidate, dedup-by-job	Avg match column, sortable
Recruiter advanced search (NEW)	Same	Match pill on each result card
Recruiter interviews list (NEW)	<code>users_job.percentage</code>	Match % badge beside name
Recruiter jobs list (NEW)	<code>COUNT(*) WHERE percentage >= 70</code>	"Top Match" column showing N high-match candidates per job
Match breakdown modal (NEW)	Lookup by <code>user_id</code>	Clickable per-job table from anywhere
Analytics dashboard	<code>AVG(uj.percentage)</code>	Avg-match progress bars
Jobseeker frontend (Next.js)	<code>profile_matched_percentage</code> from API	Green "X% Profile Match" badge (>50%)
Android app	Same	Same
Superadmin	API call or DB read	Circular progress chart

Every surface reads from the same column written by the same algorithm. No drift possible.

Algorithm version tracking — one open follow-up

`users_job.percentage` and `company_jobs_automatch.percentage` don't currently carry an `algorithm_version` column. After the rescore, every active-job row is V2. But there's no DB-level audit trail saying which algorithm version produced any given score. Suggested follow-up: add `algorithm_version TINYINT` to both tables, set to 2 on V2 writes.

Quarterly rescore as scheduled hygiene

Even with everything wired correctly, future algorithm changes will cause drift. The rescore script (`recruiter/scripts/v2-backfill.php`) is idempotent and re-runnable in one command:

```

PEM=/Users/apple/Projects/kaam-ec2/kaabil-cc-key-pair.pem
ssh -i "$PEM" ubuntu@65.1.107.228 \
    "sudo -u www-data php /var/www/tools.kaabil.cc/recruiter/scripts/v2-backfill.php --env=prod"
```

Recommended: quarterly cron, off-peak hours.

Consumer reliability — one known fragility

The `update_match_percentage` queue currently uses `noAck:true` (auto-acknowledge on receive). If the consumer crashes mid-process, that score push is silently lost. Recommend switching to `noAck:false` with manual ack-after-success and retry-on-failure, to prevent another stale cohort from accumulating after the next consumer outage. Filed but not yet implemented.

13. Where things stand today

Live on PROD right now

☑ V2 algorithm running in the ES service at `feed.kaabilprogram.org`
 ☑ Every new application gets scored by V2 (via `update_match_percentage` queue)
 ☑ Every new job posting gets Top Matches by V2 (via `user_auto_match` queue)
 ☑ All 5,659 active-on-active applications carry V2 scores
 ☑ All 153,137 active-job Top Match entries carry V2 scores
 ☑ Job edit, activate, repost all trigger V2 refresh on existing applicants
 ☑ Manual candidate-add triggers V2
 ☑ Recruiter UI shows match across: shortlist view, auto-match tab, candidate profile, kanban, global candidates list, advanced search, interviews list, jobs list (with high-match count)
 ☑ Analytics dashboard uses V2 averages
 ☑ Jobseeker frontend + Android app show V2-driven badges
 ☑ Match-breakdown modal available platform-wide (click any candidate → see per-job match table)

Backed up and reversible

☑ Full PROD MySQL dump (227 MB raw, all 241 tables)
 ☑ `users_job_pct_backup_full_20260520` (35,262 rows, in PROD DB)
 ☑ `company_jobs_automatch_backup_20260520` (153,140 rows, in PROD DB)
 ☑ One-statement rollback SQL documented per table
 ☑ Three external comprehensive reports (TSV / CSV / log archives) on disk

Open items / follow-ups

Item	Why	Effort
Switch <code>update_match_percentage</code> consumer to <code>noAck:false</code>	Prevent another stale cohort from forming if the consumer crashes	Small change in ES service config
Add <code>algorithm_version</code> <code>TINYINT</code> column to both score tables	Audit trail for which algorithm version produced any given score	DB migration + 2 INSERT/UPDATE tweaks
Quarterly rescore cron	Even with correct triggers, model changes will cause drift	Schedule the existing script
Krutika-style shortlist review at Compass Group	1 currently-shortlisted candidate with a 38-point drop — worth a sanity check (just one, not a campaign)	One conversation
Audit any cron / admin tool that flips <code>company_jobs.status</code> directly	Anything bypassing <code>JobActionService</code> won't refresh applicants	Code grep + small fix

What recruiters will notice on their next page load

- Shortlists are reordered. ~80% of candidates have higher scores; ~7% have lower
- The Auto-Match tab shows fewer "Very Strong (76+)" candidates by design — V1 was over-counting that tier
- 1,994 previously-invisible candidates are now in the actionable score range
- The new match-breakdown modal lets them click any candidate's match pill and see how that candidate scores against every job they've applied for in the recruiter's accessible companies — useful for cross-job context

Bottom line

V1 was a noisy, scattered, hard-to-tune algorithm that quietly mis-ranked the candidate pool for months. We replaced it with a 9-factor model that lives in one place, is tunable in one file, and scores every candidate the same way regardless of CV pedigree. We rescored every active-on-active application and Top Match entry, with four layers of backup, one-statement rollback per table, and zero PROD impact during the operation.

The data shows:

- **80% of candidates score higher** — V1 was systematically under-crediting most of the pool
- **611 candidates left the Very Strong tier** in Auto-Match — V1 was systematically inflating a smaller group via marquee bias
- **1,994 invisible candidates rescued** — these are the ones the recruiter ecosystem was losing
- **Only 3 currently-shortlisted candidates** saw any score drop — past hiring decisions are nearly entirely undisturbed

The recruiter's shortlist now reflects who the candidate actually is.

Generated 2026-06-03. All numbers verified against fresh PROD clone imported locally for this analysis. Names, jobs, companies, and percentages are real production data. Backups in place; rollback path is one SQL statement per table.