ChatGPT

Agentic Orchestrator & NL Criteria Architect

Overview

Finding an apartment in a competitive housing market such as **San Francisco** requires translating imprecise preferences (e.g., "quiet, sunny, not sketchy, near X, has parking, no carpet, dog-friendly, modern-ish, not a shoebox, within 30 min transit to Y, ideally under $Z but can stretch") into structured search criteria. To build an agentic apartment-hunting tool that uses OpenAI models and web-scraping (e.g., Firecrawl connectors) to search across multiple sources (Craigslist, Zillow, Realtor.com, property-management sites), we define a **Preference Spec** representation, design specialist agents, orchestrate multi-stage workflows, employ search diversity techniques, and provide explainable feedback. The tool should continually refine its search based on user feedback.

This report synthesizes best practices from research on constraint representation and search algorithms:

- **Hard vs. soft constraints** – In optimization models, *hard* constraints must be satisfied by any feasible solution, whereas *soft* constraints can be violated at a cost; soft constraints appear in objective functions as penalty terms [1]. The distinction allows users to specify "must-have" vs. "nice-to-have" features.
- **Query expansion** – Enhancing a query by adding synonyms, semantically related words, morphological variants and corrected spellings broadens recall [2]. This increases the chance of discovering listings in adjacent neighborhoods or using different terminology.
- **Exploration vs. exploitation** – The multi-armed bandit framework balances exploring new options and exploiting high-performing ones. The trade-off, widely used in machine-learning resource allocation, enables an agent to maximize value while learning unknown reward distributions [3].
- **Explainability** – Explainable recommendation systems provide transparent and interpretable reasons for why a recommendation was made [4]. Key components include transparency, interpretability, justification, user-centric design and feedback mechanisms [5]. Providing natural-language reasons for matches fosters trust and encourages engagement [6].

A. Preference Spec JSON

Representation

The **Preference Spec** captures user housing criteria in a machine-actionable structure. It separates **hard constraints** that must be met (e.g., budget ceilings, pet policy, commute limits) from **soft preferences** with weights, supports exclusions (e.g., avoid certain neighborhoods) and allows trade-offs. It also encodes commute anchors and time windows, vibe proxies (e.g., distance from busy roads for quietness) and notes missing information requiring clarification.

Category	Description	Example Fields
hard_constraints	Non-negotiable requirements; violation removes the listing. Derived from explicit statements ("must have" or safety requirements).	`max_rent`, `min_bedrooms`, `pet_policy`, `parking_required`, `commute_time_limit`, `no_carpet`, `allowed_neighborhoods`, `excluded_neighborhoods`
soft_preferences	Desired features that improve score but may be compromised. Each has a `weight` to reflect relative importance.	`sunny=0.8`, `modern_style=0.6`, `large_size=0.7`, `view`, `yard`
tradeoffs	Defines willingness to relax a soft preference to improve another. Each entry is a pair with trade-off ratios.	Example: `[{ from: "rent", to: "commute_time", ratio: 0.5 }]` means the user is willing to pay \$1 for every 2 minutes saved in commute.
commute_anchors	Locations (addresses or coordinates) and acceptable travel modes with maximum duration.	`[{ anchor: "Financial District, San Francisco", max_minutes: 30, modes: ["transit", "bike"] }]`
vibe_proxies	Transforms ambiguous “vibes” into measurable proxies using data sources.	`quiet`: distance from highways/major roads > 0.2 mi; top-floor or concrete building; language cues in listing ("quiet", "peaceful"); `not_sketchy`: crime rate below city average; lighting and safety comments in listing.
missing_info	Placeholders for information not found in listings. May trigger clarifying questions.	`natural_light_quality`, `noise_level`, `exact_square_feet`

Example Preference Spec JSON

```
{
  "user_id": "user123",
  "timestamp": "2026-01-27T12:00:00Z",
  "hard_constraints": {
    "max_rent": 3500,
    "min_bedrooms": 1,
    "pet_policy": "dog-friendly",
    "parking_required": true,
    "no_carpet": true,
    "commute_time_limit": 30,
```

```
    "commute_anchors": [
      {
        "anchor": "Financial District, San Francisco, CA",
        "max_minutes": 30,
        "modes": ["transit", "bike"]
      }
    ],
    "allowed_neighborhoods": ["Mission Bay", "SoMa", "Dogpatch", "Potrero
Hill"],
    "excluded_neighborhoods": ["Tenderloin", "Bayview"]
  },
  "soft_preferences": [
    { "feature": "quiet", "weight": 0.9, "proxies": {
"min_distance_from_major_road_m": 200, "building_type": ["concrete", "modern"],
"listing_keywords": ["quiet", "peaceful", "low-traffic"] } },
    { "feature": "sunny", "weight": 0.7, "proxies": { "south_facing": true,
"floor_to_ceiling_windows": true, "listing_keywords": ["sunny", "bright",
"natural light"] } },
    { "feature": "modern_style", "weight": 0.6, "proxies": {
"year_built_after": 2000, "renovated_after": 2015, "listing_keywords":
["modern", "updated", "remodeled"] } },
    { "feature": "large_size", "weight": 0.5, "proxies": { "min_sqft": 700 } }
  ],
  "tradeoffs": [
    { "relax": "max_rent", "benefit": "commute_time_limit", "ratio": 1.0 },
    { "relax": "quiet", "benefit": "sunny", "ratio": 0.5 }
  ],
  "vibe_proxies": {
    "quiet": { "distance_from_major_road_m": 200, "noise_level_dba": 50 },
    "not_sketchy": { "crime_rate_percentile": 50 }
  },
  "missing_info": ["exact_square_feet", "noise_level_dba",
"natural_light_quality"]
}
```

The `vibe_proxies` convert subjective notions like **quiet** into measurable criteria. For example, HUD's noise guidebook lists **near freeway auto traffic** at ~60 dB (moderate) whereas an **average residence** is around 20 dB [7]. The spec uses a noise threshold (e.g., < 50 dB inside) and requires distances > 200 m from highways to increase the chance of a quiet apartment. "Not sketchy" is proxied by local crime statistics relative to city average and by reviewing listing language for safety cues.

B. Agent Roster

Agent	Role	Responsibilities	Tools/Methods
Planner (Orchestrator)	Translates natural language requests into a Preference Spec; decides which specialist agents to invoke and orders tasks. Plans multi-pass searches, manages retries/fallbacks and triggers clarifying questions only when high value.	Uses OpenAI models (text-to-spec) and maintains user profile/history. Generates diversified search strategies and budgets (explore vs. exploit).	OpenAI API, planning algorithms, heuristics.
Query Generator	Builds multiple search queries from the Preference Spec using synonyms, neighborhood aliases and expansions.	Implements query expansion techniques [2], generates multi-source query sets (e.g., "dog-friendly modern apartment Mission Bay", "1-bedroom sunny condo SoMa parking").	Thesaurus/ontology, morphological analysis, geographic knowledge base.
Connector Agents	Specialized scrapers or API connectors for each source (Craigslist, Zillow, Realtor.com, building management sites, etc.). Each agent handles login, pagination, error recovery and extraction.	Execute search queries; parse listing data into a normalized schema (address, rent, bedrooms, amenities, text, photos, etc.); extract features via NLP (e.g., detect "quiet" in descriptions).	Firecrawl & custom scrapers; HTML parsers; schema-on-read.
Normalizer & Dedupe Agent	Cleans and deduplicates listings from multiple sources. Standardizes units (rent per month, square footage), geocodes addresses, and merges duplicates (same address or images).	Use fuzzy matching, Jaccard similarity on addresses/ photos, and handle variations (suite numbers, abbreviations).	Data-cleaning libraries, geocoding APIs.

Agent	Role	Responsibilities	Tools/Methods
Fact-Validator	Validates extracted facts (e.g., verifying commute times, pet policy, parking availability). Cross-checks with third-party data sources (Google Maps API for commute time; municipal crime data for safety).	Compares claims against external APIs; flags discrepancies or missing information.	Maps API, crime/ statistics APIs, noise maps, building records.
Scorer & Ranker	Computes a score for each listing based on hard constraints, soft preference weights, trade-offs and user feedback. Calculates penalty for soft-constraint violations [1].	Weighted scoring function; multi-armed bandit algorithm to decide exploration vs. exploitation for new neighborhoods/ sources [3].	Statistical models, reinforcement learning, ranking algorithms.
Explainability Agent	Generates plain-language explanations for why a listing matches the user's criteria and what information is missing. Follows explainability principles (transparency, interpretability, justification and user-centric design) [8].	Extracts key features (rent, distance, amenities) and cites proxies (e.g., noise level). Flags missing data and suggests clarifications.	Natural Language Generation (NLG) models, templates.
Clarification Agent	Identifies high-impact missing information or conflicting preferences and asks the user targeted clarifying questions.	Looks at `missing_info` fields and surfaces clarifications only when responses will significantly improve ranking. Explains why information is needed.	Dialog management, question-selection heuristics.
Feedback Updater	Incorporates user feedback (e.g., likes/dislikes, comments) to update weights, learn new preferences and adjust search strategies.	Implements example-critiquing loops and preference revision; updates trade-off ratios.	Active learning; user modeling.

C. Orchestration Flow

1. **Initial Parsing**

2. The user provides a natural-language query. The planner uses an LLM to parse it into a **Preference Spec** (hard constraints, soft preferences, trade-offs, commute anchors, vibe proxies).
3. If critical fields are missing (e.g., budget), the Clarification Agent asks targeted questions; otherwise, proceed.
4. **Diversified Query Generation**
5. The Query Generator expands search terms using synonyms and related words (e.g., *quiet* → "peaceful", "low-traffic"; neighborhoods synonyms like "South of Market" and "SoMa"). Query expansion increases recall but may reduce precision [2].
6. It constructs multiple query variants across price ranges, neighborhoods and sources. Some queries intentionally relax soft constraints to find near-misses.
7. **Search Plan & Exploration Budget**
8. The Planner allocates an exploration budget using a **multi-armed bandit** strategy: each arm corresponds to a neighborhood-source pair. The agent chooses between exploring under-sampled areas and exploiting high-yield ones. The bandit framework explicitly balances exploration and exploitation [3].
9. The exploration budget can be weighted by user trade-offs (e.g., allocate more budget to neighborhoods close to the commute anchor but still explore adjacent areas).
10. **Connector Execution**
11. Connector Agents execute queries on each platform. They handle pagination, login, CAPTCHAs (if legal), and scraping. They extract structured data (rent, address, bedrooms, square footage, description, photos) and store raw text for NLP.
12. Errors (e.g., site unreachable) trigger retries; after multiple failures, the Planner reallocates budget to other sources.
13. **Normalization & Deduplication**
14. The Normalizer Agent standardizes units and geocodes addresses.
15. It deduplicates listings across sources using fuzzy address matching, image hashes and contact information.
16. **Feature Extraction & Fact Validation**
17. NLP models extract features from descriptions (e.g., presence of "quiet", "modern", "no carpet") and update the listing's attribute vector.
18. The Fact-Validator uses external APIs to compute commute times to each anchor; calculates noise proxies (distance from highways and expected dB levels; noise guidebook lists near-freeway noise around 60 dB [9]); checks pet policies and parking using building data; fetches crime statistics.
19. **Scoring & Ranking**
20. For each listing, the Scorer applies hard constraints and discards any violating listing.
21. It computes a weighted score: sum of soft preference weights minus penalties for soft-constraint violations [1]. Trade-offs adjust scores based on user willingness to pay more for a shorter commute.
22. The bandit model updates arm values based on observed quality (e.g., number of high-scoring listings returned) and adjusts exploration/exploitation for subsequent searches.
23. **Explainability & Presentation**
24. The Explainability Agent generates natural-language explanations for top listings. It follows explainability principles (transparency, interpretability, justification and user-centric design) [10]. For example, a listing is recommended because it is under budget, 20 minutes from the Financial District via transit, includes parking, allows dogs, and is in a quiet area (0.3 mile from highways, expected noise < 50 dB).

25. It also highlights missing information (`missing_info`) and suggests contacting the landlord for details or scheduling an in-person visit.
26. **User Feedback & Clarification**
27. The system presents ranked listings with explanations. The user can like/dislike, adjust weights or provide comments.
28. The Feedback Updater updates the Preference Spec accordingly. It uses example-critiquing to refine preferences and can adjust weights or trade-offs.
29. When user feedback indicates confusion (e.g., the term “modern-ish” could mean anything built after 2000 or simply renovated), the Clarification Agent asks targeted questions to refine proxies.
30. **Iterative Search**
    1. The Planner triggers another search cycle with updated preferences and exploration budgets.
    2. The system continues iterating until the user is satisfied or the search space is exhausted.

D. Search Diversity / Robustness Techniques

1. **Query Expansion & Synonym Mining** – Use thesaurus-based and corpus-based methods to find synonyms, hyponyms and morphological variants [2]. Expand neighborhoods using local slang (e.g., “South of Market” vs. “SoMa”) and nearby areas; expand features ("quiet" → “peaceful”, “low-traffic”; “parking” → “garage”, “off-street”).

2. **Multi-Armed Bandit Exploration** – Model each neighborhood-source pair as an arm. Initially allocate equal sampling. After each round, update estimated reward (average listing score) and allocate exploration probability using strategies like **epsilon-greedy** or **Thompson Sampling**. The bandit framework ensures that the agent both explores new areas and exploits high-yield sources [3].

3. **Breadth-First Sweeps & Deep Dives** – Start with broad sweeps across many neighborhoods and sources (lower ranking threshold). Identify clusters of promising results (e.g., Mission Bay units with high scores) and perform deep dives using more specific queries and additional keywords.

4. **Counterfactual Searches (Near-Miss Analysis)** – For each soft constraint, intentionally relax it (e.g., allow noise up to 65 dB; allow 35-minute commute) and search for listings that would have been filtered out. Present these near-miss listings separately as “stretch options,” showing what trade-off would be necessary (e.g., pay \$200 more for 5 minutes less commute). This helps uncover hidden gems.

5. **Missed Listing Detector** – Monitor distribution of results across neighborhoods, price ranges and attributes. If certain areas yield zero results, run targeted sweeps: broaden search radius, increase budget slightly, or search alternative sources. For example, if there are no dog-friendly listings under \$3,500 in Potrero Hill, the agent temporarily increases the budget to \$3,800 or includes adjacent neighborhoods. A threshold triggers the system to report to the user that constraints may be too tight.

6. **Temporal & Source Diversification** – Stagger searches across times of day to capture newly posted listings. Use varied sources (official property-management websites, rental platforms, social media groups) to avoid missing listings that appear only on a specific platform.

E. Example Run Walkthrough

User request: “Looking for a quiet, sunny 1-bedroom near Mission Bay. Needs parking, no carpet, dog-friendly, modern-ish (but not a shoebox), within 30 minutes by transit to the Financial District. Ideally under $3,500 but I can stretch if necessary.”

Step 1 – Parsing into Preference Spec

The Planner uses an LLM to extract constraints:

- **Hard constraints**: `max_rent: 3500`, `min_bedrooms: 1`, `pet_policy: dog-friendly`, `parking_required: true`, `no_carpet: true`, `commute_time_limit: 30` minutes to “Financial District”, allowed neighborhoods include Mission Bay and adjacent areas; exclude high-crime areas (via general user safety preference).
- **Soft preferences** (with weights): `quiet (0.9)`, `sunny (0.7)`, `modern_style (0.6)`, `large_size (0.5)`. A vibe proxy for “not a shoebox” sets `min_sqft` of 700.
- **Trade-offs**: user is willing to increase rent to reduce commute time (`ratio 1.0`).

Step 2 – Diversified Query Generation

The Query Generator expands “quiet” to synonyms like “peaceful”, “low-traffic” and “quiet building”; expands “modern-ish” to “remodeled”, “updated” and “newer building”; includes neighborhood aliases (“Mission Bay”, “Mission Creek”, “South Beach”). It generates queries such as:

1. `"1 bedroom apartment Mission Bay dog friendly parking modern updated"`
2. `"quiet condo near Mission Creek sunny no carpet dog OK"`
3. `"SoMa 1BR modern parking bright"` (expansion to adjacent SoMa)

Step 3 – Search Plan & Exploration Budget

The multi-armed bandit allocates equal budget to arms like (Mission Bay, Craigslist), (Mission Bay, Zillow), (SoMa, Realtor.com). After each round, the agent updates expected rewards. If SoMa listings score high, more budget is directed there.

Step 4 – Connector Execution

Connector Agents crawl each source. Suppose they find 50 listings across four neighborhoods. Raw data includes descriptions, photos and metadata.

Step 5 – Normalization & Deduplication

The Normalizer standardizes rents, geocodes addresses and merges duplicates (two websites listing the same building). After deduplication, 35 unique listings remain.

Step 6 – Feature Extraction & Fact Validation

NLP models mark each listing for features. For example, one listing describes a "quiet, top-floor condo with floor-to-ceiling windows, parking included." The Fact-Validator computes a 22-minute transit commute to the Financial District; noise proxies show the building is 0.3 mile from the nearest freeway, expected noise ~45 dB (below 50 dB threshold based on HUD guide [7]). Pet policy confirms dogs allowed; building is built in 2015.

Step 7 – Scoring & Ranking

Hard constraints filter out 10 listings lacking parking or allowing only cats. The Scorer computes scores for the remaining 25. The top listing (above) scores 0.85: high on quiet and sunny, meets modern-style, large square footage (750 sq ft), rent \$3,450. A second listing is $3,200 but only 650 sq ft; it is still considered because the user can stretch on size.

Step 8 – Explanation & Presentation

For the top listing, the Explainability Agent outputs:

> **Why this matches:** This 1-BR condo is \$3,450/month, within your budget and only 22 minutes by transit to the Financial District. It has dedicated parking and allows dogs. The listing notes a *quiet top-floor* unit with *floor-to-ceiling windows* ("sunny") and no carpet. Built in 2015, it meets your *modern-ish* preference. The building is ~0.3 mile from the nearest freeway, so estimated noise levels are about 45 dB (quiet residential levels [7]). The unit is 750 sq ft, so it isn't a shoebox.
>
> **What's missing/uncertain:** The listing doesn't specify the exact noise insulation or whether there is a balcony. You might ask about natural light orientation and confirm no carpet.

Step 9 – Feedback & Clarification

The user likes the top listing but mentions "modern-ish" means architecture after 2010, not just renovations. The Clarification Agent updates the modern-style proxy to `year_built_after: 2010` and increases its weight. The bandit model updates exploration budgets. The cycle repeats.

Conclusion

The proposed **Preference Spec** structure, agent roster and orchestration flow enable an agentic system to transform messy apartment criteria into structured constraints and preferences, perform diversified searches across multiple sources, and iteratively refine results. By employing query expansion and multi-armed bandit exploration, the system uncovers both obvious and hidden gems while balancing search breadth and depth. Explainability and targeted clarifications ensure the user understands why each listing was selected, improving trust and enabling faster decision-making.

---

[1] optimization - Soft constraints and hard constraints - Operations Research Stack Exchange
https://or.stackexchange.com/questions/1050/soft-constraints-and-hard-constraints

2 Query expansion - Wikipedia
https://en.wikipedia.org/wiki/Query_expansion

3 Multi-armed bandit - Wikipedia
https://en.wikipedia.org/wiki/Multi-armed_bandit

4 5 6 8 10 Explainability In Recommendation Systems
https://www.meegle.com/en_us/topics/recommendation-algorithms/explainability-in-recommendation-systems

7 9 THE NOISE GUIDEBOOK
https://www.huduser.gov/portal/portal/sites/default/files/pdf/The-Noise-Guidebook.pdf