Tarkov Weapon Mod Optimizer

Automated Build Optimization using Constraint Programming

Imposs1ble嗨

January 4, 2026

The Problem: Complexity in Escape from Tarkov

Escape from Tarkov features one of the most complex weapon modification systems in gaming.

Challenges for New Players:

- **Overwhelming scale:** 2,500+ mods, each affecting recoil, ergonomics, weight, accuracy
- **Nested compatibility:** Mods attach to mods (rail → mount → sight)
- **Combinatorial explosion:** A single AR-15 has 50+ slots, millions of valid builds
- **Non-obvious trade-offs:** Is a 50k grip worth it over a 5k one?
- **Dynamic availability:** Prices change; items locked behind trader levels, quests, and flea market access

Result: Players spend hours theory-crafting or blindly copy "meta builds" without understanding trade-offs.

The Solution: Automated Optimization

Project Goal: Create a tool that mathematically guarantees the "best" weapon build for a given set of constraints.

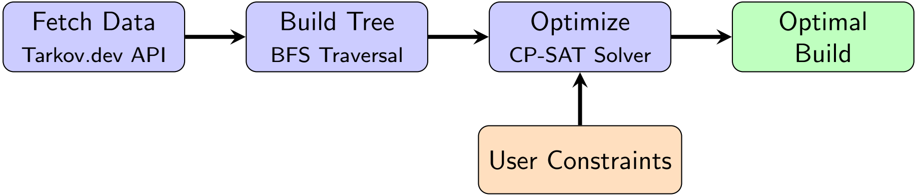
Use Cases:

- *"I have 200k roubles—what's the best M4 I can build?"*
- *"What's the absolute lowest recoil AK, money no object?"*
- *"I'm level 15 with LL2 traders—what can I actually buy?"*
- *"Is spending 100k more actually worth it?"*
- *"Show me all viable builds on a budget curve"*

Key Capabilities:

- **Multi-objective**: Balance recoil, ergonomics, and price
- **Budget-aware**: Hard limit or soft penalty on cost
- **Trader filtering**: Respect loyalty levels (LL1–LL4)
- **Flea market toggle**: Include or exclude market prices
- **Pareto frontier**: Visualize price vs. performance trade-offs

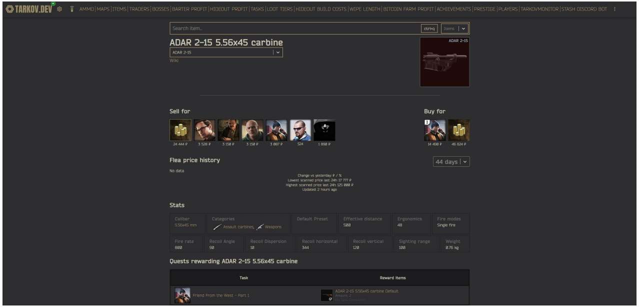
Pipeline Overview



Fetching Data

Tarkov.dev GraphQL API

- Queries all guns, mods, and compatibility rules
- **Up-to-date prices** from traders and flea market
- Trader loyalty level requirements
- Real-time flea market prices
- Local caching for speed


TARKOV.DEV
AMMO | MAPS | ITEMS | TRADERS | BOSSES | BARTER PROFIT | HIDEOUT PROFIT | TASKS | LOOT TIERS | HIDEOUT BUILD COSTS | WIPE LENGTH | BITCOIN FARM PROFIT | ACHIEVEMENTS | PRESTIGE | PLAYERS | TARKOVMONITOR | STASH DISCORD BOT
Search item..
ctrl+q
Items
ADAR 2-15
Wiki
I

ADAR 2-15 5.56x45 carbine

Sell for: 24 444 ₽, 3 526 ₽, 3 150 ₽, 3 150 ₽, 3 087 ₽, $24, 1 890 ₽

Buy for: 14 490 ₽, 46 624 ₽

Flea price history — 44 days

No data

Change vs yesterday: ₽ / %
Lowest scanned price last 24h: 17 777 ₽
Highest scanned price last 24h: 125 000 ₽
Updated: 2 hours ago

Stats

Caliber	Categories	Default Preset	Effective distance	Ergonomics	Fire modes
5.56x45 mm	Assault carbines, Weapons		500	40	Single fire

Fire rate	Recoil Angle	Recoil Dispersion	Recoil horizontal	Recoil vertical	Sighting range	Weight
800	90	10	344	120	100	0.76 kg

Quests rewarding ADAR 2-15 5.56x45 carbine

Task	Reward items
Friend From the West - Part 1	ADAR 2-15 5.56x45 carbine Default Amount: 2 On Task Completion

Data Processing: From API to Item Lookup

Step 1: Raw API Data Normalization

Slot Extraction Challenges:

- Guns and mods have different property structures
- `filters.allowedItems` can be:
  - List of objects: `[{"id": "abc"}]`
  - List of strings: `["abc", "def"]`
- Must normalize to flat list of IDs

Price Validation:

- Mods without valid `buyFor` offers are filtered out
- Prevents selecting items that can't actually be purchased

Stat Normalization:

- **Recoil modifier** formats differ:
  - Top-level: integer % (e.g., -5)
  - Properties: decimal (e.g., `-0.05`)
- Normalize all to decimal format
- Handle null/missing values gracefully

Conflict Extraction:

- `conflictingItems`: Items that can't coexist
- `conflictingSlotIds`: Slots blocked by this item

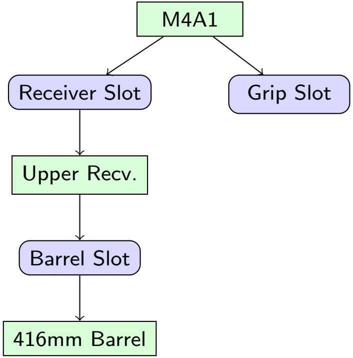
Building the Compatibility Tree

BFS Process:

1. Start from the base weapon's slots
2. For each slot, find all allowed items
3. Add those items to the queue
4. For each item, discover *its* slots
5. Repeat until no new items found

Data Structures Built:

- `slot_items`: Slot → [compatible items]
- `slot_owner`: Slot → parent item
- `item_to_slots`: Item → [owned slots]



Items own slots; slots contain items.

Introduction to CP-SAT Solver

CP-SAT = **C**onstraint **P**rogramming + **SAT** (Boolean Satisfiability)

What is it?

A solver from Google's OR-Tools library that finds optimal solutions to combinatorial problems by combining:

- **Constraint Programming**: Express complex rules declaratively
- **SAT Solving**: Efficient Boolean logic reasoning
- **Linear Programming**: Optimize numerical objectives

Why CP-SAT for this problem?

- Handles Boolean decisions naturally (select mod or not)
- Scales to thousands of variables and constraints
- Guarantees optimal solution (not just "good enough")
- Solves in milliseconds for typical weapon builds

How CP-SAT Works

1. Model Definition

- Create Boolean variables: $X_i \in \{0, 1\}$
- Add constraints (rules)
- Define objective function

2. Solving Process

- Propagation: Infer forced values
- Search: Branch on undecided variables
- Backtrack: Undo bad decisions
- Prune: Skip impossible branches

3. Key Techniques

- *Unit Propagation*: If a constraint forces a value, apply it immediately
- *Conflict Learning*: Remember why branches failed to avoid repeating mistakes
- *Bound Tightening*: Use objective to prune suboptimal branches early

Result: Explores the search space efficiently, proving optimality.

Applying CP-SAT to Weapon Builds

Core Idea: Define what a valid build looks like, let the solver find the best one.

The Three Components

1. **Variables**: Boolean X_i for each reachable mod ($X_i = 1$ if selected)
2. **Constraints**: Compatibility rules, budget limits, required slots
3. **Objective**: Maximize ergonomics, minimize recoil and price

The solver explores millions of mod combinations, pruning invalid builds instantly, and returns the mathematically optimal configuration.

Constraint 1: Slot Mutex (One Item Per Slot)

Rule: Each slot can hold at most one item.

Formula

For each slot s with candidate items $\{i_1, i_2, ..., i_n\}$:

$$\sum_{i \in \text{slot}_s} X_i \leq 1$$

where $X_i = 1$ if item i is selected, 0 otherwise.

Example: A pistol grip slot can have a Zenit RK-3 *or* an Ergo PSG-1, but not both simultaneously.

Constraint 2: Parent Dependency

Rule: A mod can only be selected if its parent slot owner is also selected.

Formula

If item c (child) fits in a slot owned by item p (parent):

$$X_c \leq X_p$$

A child cannot be selected without its parent.

Example: You cannot attach a foregrip to a handguard unless that handguard is installed on the weapon.

$$Gun \rightarrow Handguard \rightarrow Foregrip$$

Constraint 3: Conflicting Items

Rule: Some items cannot be used together (API-defined conflicts).

Formula

For each conflict pair (a, b) from the API:

$$X_a + X_b \leq 1$$

At most one of the conflicting items can be selected.

Example: Certain dust covers conflict with specific charging handles due to physical interference—the game prevents mounting both.

Constraint 4: Required Slots

Rule: Certain slots *must* be filled for a functional weapon.

Formula

For each required slot s:

$$\sum_{i \in \text{slot}_s} X_i \geq 1$$

At least one compatible item must fill the slot.

Vital Slots (AR-15 platform):

- Barrel, Gas Block, Pistol Grip, Charging Handle, Receiver, Handguard

Without these, the weapon cannot function in-game.

Constraint 5: Budget Limit

Rule: Total build cost must not exceed the player's budget.

Formula

$$\sum_i \text{Price}_i \cdot X_i \leq \text{Budget}$$

The sum of all selected item prices must stay within budget.

Smart Pricing:

- Considers trader loyalty levels (LL1–LL4)
- Checks flea market availability
- Automatically finds the cheapest source for each item

The Objective Function

Goal: Find the build that best matches the player's priorities.

Weighted Objective (Maximize)

$$\text{Score} = W_E \cdot \text{Ergo} - W_R \cdot \text{Recoil} - W_P \cdot \text{Price}$$

- W_E: Ergonomics weight (higher = faster ADS, less stamina drain)
- W_R: Recoil weight (lower = more controllable)
- W_P: Price weight (lower = cheaper build)

Example Profiles:

- *Meta Build*: $W_R = 100, W_E = 0.5, W_P = 0$
- *Budget Build*: $W_R = 1, W_E = 1, W_P = 0.5$

How Stats Are Calculated

Ergonomics (Additive):

$$\text{Ergo}_{\text{final}} = \text{Ergo}_{\text{base}} + \sum_i \text{Ergo}_i$$

Simply add up all mod bonuses.

Soft-capped at 100 in-game.

Recoil (Multiplicative):

$$\text{Recoil}_{\text{final}} = \text{Recoil}_{\text{base}} \times \left(1 + \sum_i r_i\right)$$

Where r_i is the percentage modifier (e.g., $-0.05 = -5\%$).

Stacking suppressors gives diminishing returns.

Gunsmith Task Solver

Problem: Gunsmith quests require building weapons to exact specifications.

Task Definition (JSON):

- Target weapon name
- Stat constraints:
  - Min ergonomics, max recoil
  - Min magazine capacity
  - Max weight, min sighting range
- Required specific items
- Required categories (e.g., "Silencer")

How It Works:

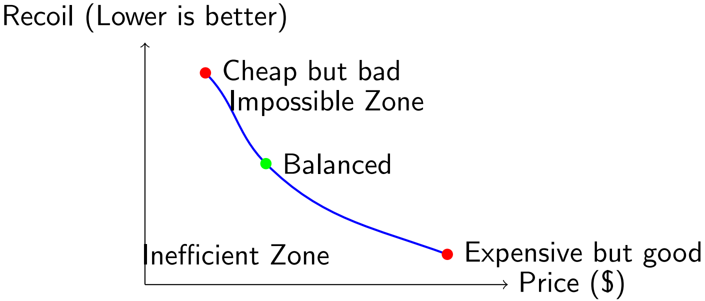
1. Load task from `tasks.json`
2. Convert requirements to constraints:
   - Required items → $X_i = 1$
   - Categories → $\sum_{i \in \text{cat}} X_i \geq 1$
   - Stats → hard bounds
3. Objective: minimize price
4. Solver finds cheapest valid build

Result: Automatically solves all 25 Gunsmith quests with optimal (cheapest) builds.

Pareto Frontier Analysis

What is it?

- A curve showing the trade-off between two conflicting goals (e.g., Price vs. Recoil).



Why use it? Allows users to see if spending an extra 50k Roubles actually gives a significant benefit.

Acknowledgements

Data Source

TARKOV.DEV

Open-source GraphQL API for Escape from Tarkov

AI Coding Assistants

Claude Code
Anthropic

Gemini CLI
Google

MiniMax
MiniMax

All the AI assistants helped me with code generation, debugging, and documentation