# YakRobot Protocols Stack

*Coordinating robots.*

An open-source protocol stack for decentralized, distributed robotics — identity, command, marketplace, attestation and fleet security.

Distributed Robotics Group · Protocol Institute · yakrobot.com

THE THESIS

# Coordination is a protocol problem

Building one robot is an engineering challenge. Getting **two or more to coordinate** is a protocol problem.

### AI Agents

LLMs that perceive and act on the world

### Blockchain

neutral identity, settlement & attestation

### Robotics

real hardware doing real physical work

*Protocols are engineered arguments — so we argue them, then test them on real robots.*

WHAT COORDINATION NEEDS

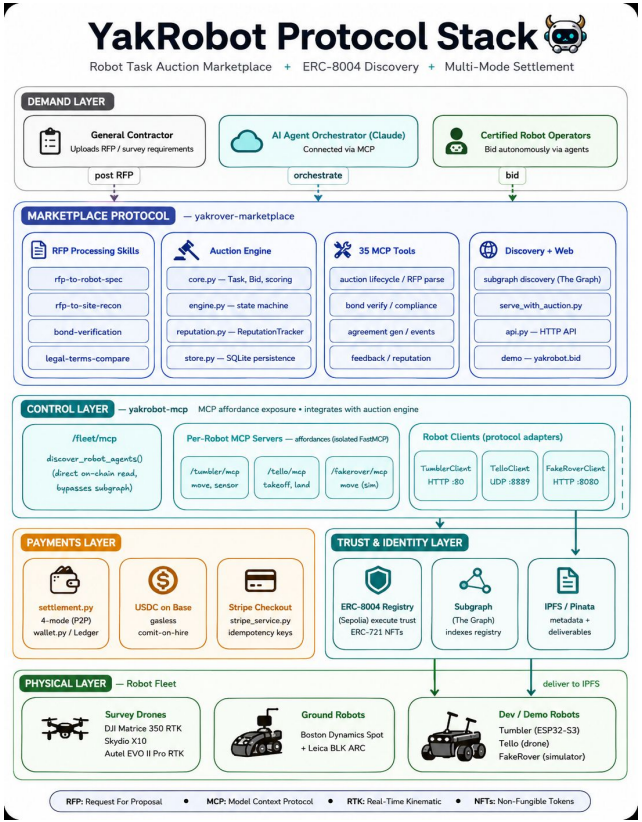
# Five things a robot must do with others

| | |
|---|---|
| **Identity** | prove who it is, verifiably |
| **Command** | accept work in a common language |
| **Payment** | get paid for work done |
| **Attestation** | prove what it actually did |
| **Trust / custody** | operate safely in a fleet |

*Each becomes a layer. None of them is "the robot" — they're the connective tissue between robots.*

THE FULL STACK

# We climb it from the ground up




**6 — Demand Layer**
who brings work & capacity

**5 — Marketplace Protocol**
post jobs, robots bid

**4 — Control Layer**
MCP gateway commands robots

**3 — Payments Layer**
settlement: USDC · Stripe · ledger

**2 — Trust & Identity Layer**
ERC-8004 · subgraph · IPFS

**1 — Physical Layer**
the robot fleet

*↑ we present bottom-to-top*

DESIGN PRINCIPLE

# Composable, not monolithic

No all-or-nothing platform. Six building blocks — each a separate protocol, useful on its own, snapping together through open standards. A robot speaks only what it uses.

**ERC-8004**

on-chain identity & discovery

**MCP**

command surface for any LLM

**x402 · USDC · Stripe**

pay-per-use settlement

**KZG / EAS**

sensor attestation & proofs

**Safe{Wallet}**

multi-sig fleet custody

**ESP32-S3/RPi**

open hardware platform

*Open standards as the glue — permissionless to extend, impossible to capture.*

LAYER 1 OF 6

# Physical Layer

1

The robot fleet — protocols are only real if they run on real hardware.

# A transport-agnostic fleet

Drones, ground robots, and dev rovers all join the same way — the stack doesn't care how a robot talks, only that it can.

| | |
|---|---|
| **Survey Drones** | DJI Matrice 350 RTK · Skydio X10 · Autel EVO II Pro RTK |
| **Ground Robots** | Boston Dynamics Spot + Leica BLK ARC |
| **Dev / Demo Robots** | Tumbller (ESP32-S3) · Tello (drone) · FakeRover (sim) |

# ESP32-S3 Tumbller

Our first reference platform: a cheap, hackable, self-balancing rover. Anyone can build one and put it on the network.

| | |
|---|---|
| **Compute** | ESP32-S3 motion MCU + ESP32-CAM for vision |
| **Motion** | SimpleFOC field-oriented control · self-balancing diff-drive |
| **Identity** | ATECC608B secure element — hardware root of trust (in design) |
| **Open** | schematics + firmware fully open-source |

*More platforms — arms, larger rovers — are being integrated next.*

LAYER 2 OF 6

# Trust & Identity

2

Who the robot is — a verifiable, decentralized identity, discoverable by anyone.

# ERC-8004 robot identity

Every robot is a registered agent on-chain — a verifiable identity that isn't owned by any platform.

| | |
|---|---|
| **On-chain registry** | ERC-721 agent NFTs on Sepolia / Base, one per robot |
| **Agent card on IPFS** | endpoint, tool list, bidding_terms — pinned via Pinata |
| **Hardware root (in design)** | ATECC608B signs every request with a non-extractable key |

# Discovery without a directory

Robots find each other by reading the same registry they registered in. No central matchmaker to trust, query, or take down.

- **Direct on-chain read**

  discover_robot_agents() reads ERC-8004 directly — bypasses any index

- **Subgraph (The Graph)**

  indexes the registry for fast GraphQL queries when you want them

*Two paths, same source of truth — the chain is canonical; the subgraph is a convenience.*

# Proving honesty & securing the fleet

## ZK Sensor Attestation

IN DESIGN

KZG polynomial commitments batch-attest sensor data. Provably honest robots — without revealing raw telemetry. Turns “trust me” into “verify me.”

## 2-of-3 Fleet Protocol

IN DESIGN

Multi-sig custody on Safe{Wallet}: operator, hardware and recovery keys. Any two sign — a lost laptop or stolen robot never compromises the swarm.

LAYER 3 OF 6

# Payments

3

How robots get paid — settlement triggered by verified completion, not invoices.

# Card, stablecoin, or x402

The buyer picks the rail; the robot gets paid the same way. Fiat for today's operators, crypto-native for autonomous agents.

| | |
|---|---|
| **Stripe** | card checkout · fiat on-ramp · webhook-confirmed |
| **USDC on Base** | stablecoin · gasless · pays the robot's wallet directly |
| **x402** | HTTP-native pay-per-use · agents settle micropayments inline |

*Whichever rail, payment releases on verified completion — escrow in, hold on dispute.*

# Human-in-the-loop, or autonomous

### Mode A — Human-in-the-loop

```
requires_approval = True (default)
```

Task posted → robots bid → bid accepted → buyer pays → operator calls fleet_execute_task to approve → robot executes.

### Mode B — Autonomous

```
requires_approval = False
```

Same first steps, then the webhook fires → on_payment_confirmed() schedules execution automatically — no human in the loop.

LAYER 4 OF 6

# Control Layer

4

How robots are commanded — the MCP gateway.

# One MCP gateway, one tunnel

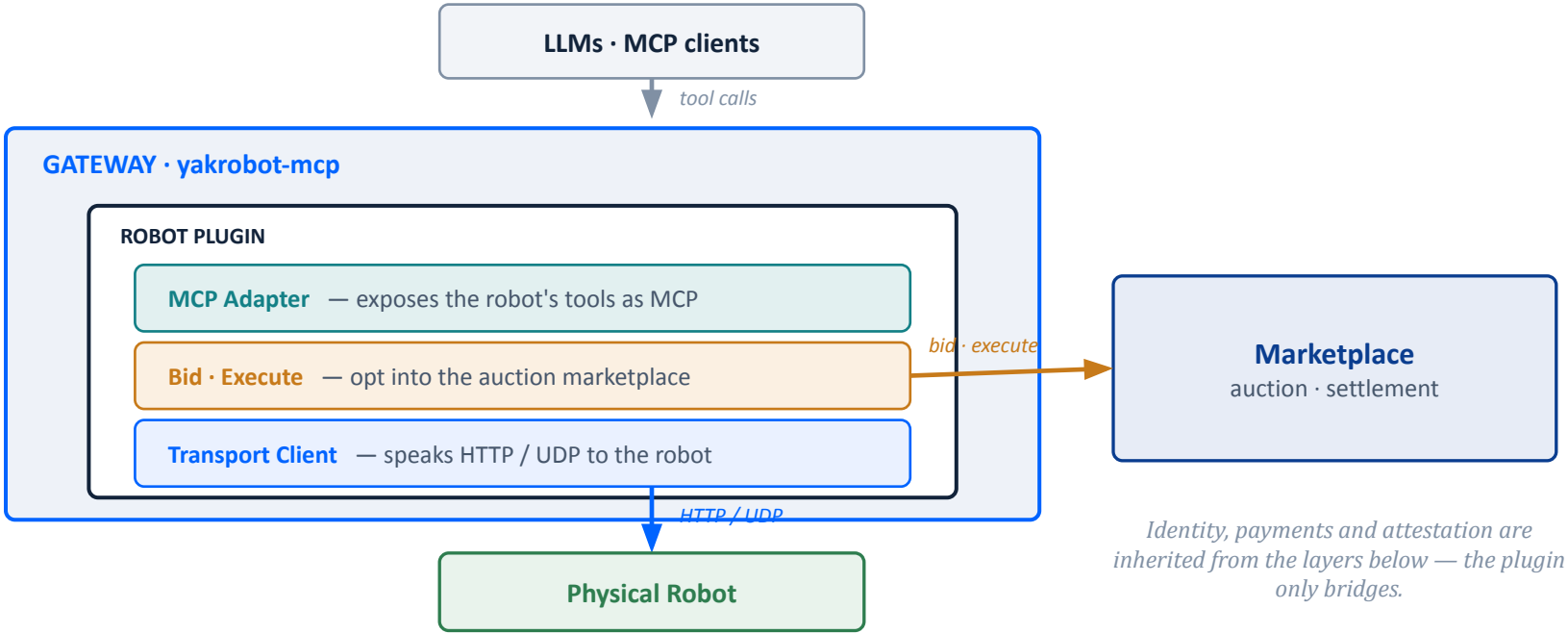
yakrobot-mcp is a FastAPI + MCP gateway. LLMs **discover, command and query robots in natural language** — each robot behind its own isolated MCP instance.

- `/fleet/mcp` — fleet tools + on-chain discovery
- `Per-robot MCP` — isolated FastMCP server per robot
- `Robot Clients` — protocol adapters (HTTP / UDP)
- `One ngrok URL` — single port fronts everything

*Add a robot, get an endpoint — the gateway consolidates all the shared infrastructure.*

# How a robot plugs into yakrobot-mcp gateway

A robot joins as a plugin that slots into the gateway. The plugin carries three faces: an MCP adapter toward the LLM, a transport client toward the hardware, and bid/execute hooks toward the marketplace.




*Identity, payments and attestation are inherited from the layers below — the plugin only bridges.*

# Three robots, three adapters

### Tumbller

`ESP32-S3 · HTTP :80 · httpx`

`tumbller_move · tumbller_is_online · tumbller_get_temperature_humidity`

`+ robot_submit_bid · robot_execute_task · robot_get_pricing`

### Tello

`DJI drone · UDP :8889 · djitellopy`

`tello_takeoff · tello_land · tello_move · tello_rotate · tello_flip`

`+ robot_submit_bid · robot_execute_task · robot_get_pricing`

### FakeRover

`simulator · HTTP :8080`

`move (sim) — exercises the full loop with no hardware`

`+ robot_submit_bid · robot_execute_task · robot_get_pricing`

# From LLM to motor, and back

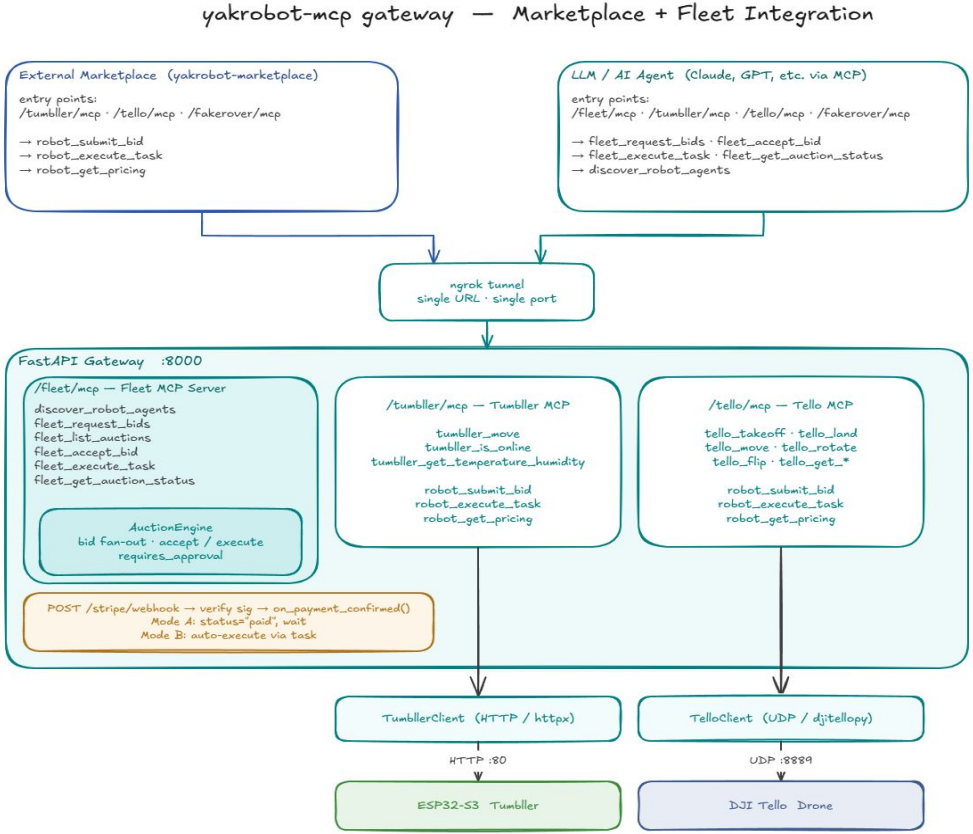
1 LLM / agent calls a tool on `/fleet/mcp` or a robot

2 Gateway routes to that robot's isolated MCP server

3 Client adapter speaks the robot's transport

4 Robot works, returns `delivery_data` JSON

5 Payment settles — USDC or Stripe



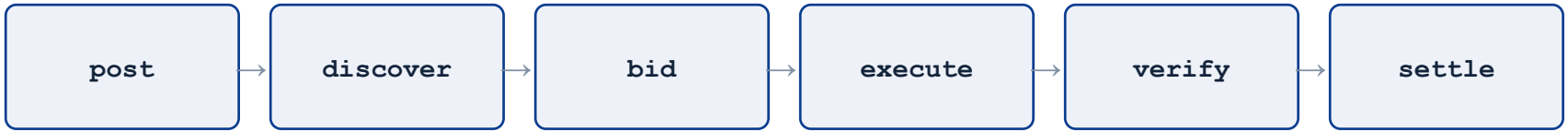
LAYER 5 OF 6

# Marketplace Protocol

5

Matching work to robots — one composable layer, built on everything below.

# Post → bid → execute → settle




**Composes the stack:** identity (who can bid) · control (how work runs) · attestation (proof of result) · payments (settlement).

- **Live demo — yakrobot.bid/demo**
  100 robots · 18 operators · 10 RFP presets · EAS attestation · geographic filtering

# Inside yakrover-marketplace

Four blocks turn the lower layers into a working auction. Here's what each is for — then what's inside.

| | |
|---|---|
| **RFP Processing** | turns a contractor's RFP into machine-readable task specs |
| **Auction Engine** | runs the auction — tasks, bids, scoring, state machine |
| **MCP Tools** | the agent-facing surface — 42 tools over the protocol |
| **Discovery + Web** | find operators on-chain · serve the demo & API |

*All MIT-licensed protocol code, in the auction/ package.*

# The engine room

## Auction Engine

`auction/`

- **`core.py`**
  Task, Bid, scoring, haversine geo
- **`engine.py`**
  state machine · geo + busy filtering
- **`reputation.py`**
  operator scoring & history
- **`store.py`**
  SQLite persistence

## RFP Processing

`skills + auction/`

- **`rfp-to-robot-spec`**
  RFP → structured task specs
- **`rfp-to-site-recon`**
  RFP → site context
- **`bond-verification`**
  Treasury Circular 570 checks
- **`legal-terms-compare`**
  contract / ConsensusDocs diff

# Tools & reach

## MCP Tools · 42

`auction/mcp_tools.py`

- **`auction lifecycle`**
  post · bid · accept · execute · settle
- **`RFP + operators`**
  parse RFPs · register operators
- **`compliance & bonds`**
  Part 107 · PLS · COI · bonds
- **`attestation + feedback`**
  EAS attestation · reputation

## Discovery + Web

`demo/ · worker/ · infra/`

- **`subgraph`**
  operator discovery (The Graph)
- **`gateway`**
  MCP server (mcp_server.py)
- **`api`**
  HTTP API · web frontend
- **`demo`**
  yakrobot.bid · explorer · landing

LAYER 6 OF 6

# Demand Layer

6

Who brings the work — two sides of a market meeting at a structured RFP.

# Supply meets demand at an RFP

### General Contractor

uploads RFP / survey requirements

`post RFP`

### AI Agent Orchestrator

Claude, connected via MCP

`orchestrate`

### Certified Robot Operators

bid autonomously via agents

`bid`

Everything beneath this layer is protocol. The top is just humans and their agents asking for work to be done.

THE FULL CLIMB

# Six layers, one composable stack

| | |
|---|---|
| **6 · Demand** | contractors + agents bring work |
| **5 · Marketplace** | post → bid → execute → settle |
| **4 · Control** | MCP gateway commands robots |
| **3 · Payments** | USDC · Stripe · 4-mode ledger |
| **2 · Trust & Identity** | ERC-8004 · subgraph · IPFS |
| **1 · Physical** | the robot fleet on the ground |

LIVE

# Demo: the stack on real robots

| | |
|---|---|
| **Discover** | an LLM finds a robot through its ERC-8004 identity |
| **Command** | drive a real robot in natural language via the gateway |
| **Auction** | post a task; robots bid, win, execute, return data |

*Videos at yakrobot.com/#platform — or run the live auction at yakrobot.bid/demo.*

THE NETWORK

# Open protocols, real robots

| 6 | 5 | 1 |
| --- | --- | --- |
| composable protocols | robot sites online | shared on-chain registry |

Every robot registers a verifiable identity on-chain, then discovers its peers through the same registry — no central operator, no single point of failure.

**All code, hardware and protocols — open source.** yakrobot.com · yakrobot.bid · github.com/YakRoboticsGarage · Distributed Robotics Group, Protocol Institute