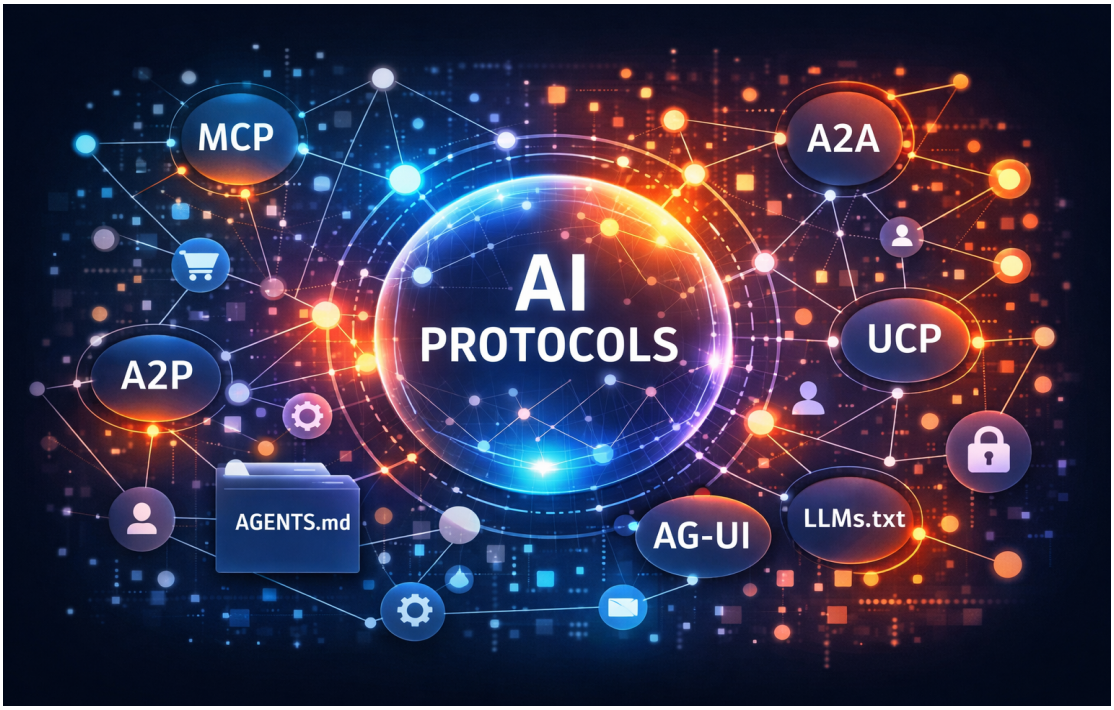
Agentic AI Protocol Landscape

(as of 21st Jan 2026)



Agentic AI systems are no longer isolated chatbots — they plan, collaborate and execute workflows across networks of tools, agents, users and payment rails. This explosion in autonomy has spawned a proliferation of **protocols** aimed at standardizing communication, context sharing, user interaction and commerce. This document provides a **comprehensive inventory** of the latest agentic AI–related protocols and specifications. Each entry describes what the protocol does, how it works, who created it, its benefits, limitations and key use cases. The protocols are grouped by thematic area (context, inter-agent communication, payments & commerce, UI, file formats, identity & networks, historical standards, etc.).

This inventory should serve as a **reference guide** for anyone building or studying agentic systems. As the field evolves, staying informed about emerging protocols and understanding their interplay will be crucial for designing interoperable, secure and effective AI agents.

The Next Platform War is not Model vs Model.
It's Protocol vs Protocol.

1 Classification of agentic protocols

Agentic protocols can be broadly organised into several layers:

- **Context and tooling access** – protocols that allow agents to discover and call external tools or incorporate extra knowledge. Examples: Model Context Protocol (MCP), Agent Skills/Skill.md.
- **Agent–agent communication** – protocols for agents to discover, authenticate and interact with each other. Examples: Google's A2A, Agent Communication Protocol (ACP) by AGNTCY, Agent Protocol, AITP and the research-oriented Agora protocol.
- **Commerce & payments** – standards enabling agents to shop and handle financial transactions. Examples: AP2/A2P, Agentic Commerce Protocol (OpenAI/Stripe), Universal Commerce Protocol (UCP), Trusted Agent Protocol (Visa), Agent Pay (Mastercard) and x402 micropayments.
- **User interfaces** – protocols describing how agents interact with human users and generate UIs. Examples: AG-UI, Google's A2UI and Open-JSON-UI.
- **File-based guidance and skills** – lightweight specifications that provide agents with instructions and capabilities. Examples: AGENTS.md, Claude.md, llms.txt, SKILL.md and Agents.json.
- **Identity & network architecture** – protocols for decentralized identity, negotiation and multi-agent networking. Examples: Agentic JWT (A-JWT), Agent Network Protocol (ANP), Summoner/SPLT, Agora and the Language Model Operating System (LMOS).

2 Context and tooling access

2.1 Model Context Protocol (MCP)

Publisher/Date: Anthropic, November 2024.
Purpose: Provide a simple way for language-model agents to discover and use external tools (APIs, file systems, databases) and inject relevant context into their prompts (source: https://modelcontextprotocol.io/).

How it works: Agents call an MCP server to retrieve a registry of available tools. Each tool is described with schemas and capabilities, allowing the agent to call it via structured requests. MCP abstracts away underlying transport and security, so a tool can be swapped without retraining the agent. Features include capability discovery, structured context schemas, tool abstraction and decoupled extensibility.

Use cases: Query internal knowledge bases, call enterprise APIs, browse websites, access databases and automate developer workflows. MCP is widely used in GitHub Copilot and other coding assistants.

Pros: Simplifies tool integration; standardizes context injection; decouples agents from specific API implementations; widely adopted.
Cons: Focuses on tool invocation, not inter-agent coordination; requires hosting an MCP server; still evolving.

Analogy: Think of MCP as a "plugin ecosystem" similar to browser extensions — it tells an agent what capabilities exist and how to invoke them, without the agent needing to understand the underlying API implementation.

2.2 Agent Skills and SKILL.md

Publisher/Date: Originally Anthropic; adopted by GitHub and others (2024–2025).
Purpose: Package procedural knowledge and resources so agents can load domain-specific skills on demand. A skill is a directory containing scripts, models and a `SKILL.md` file with YAML front-matter describing the skill name, version and instructions. Skills are stored in project directories or global locations and can be version-controlled (source: https://agentskills.io/).

Use cases: Teach agents how to perform tasks (e.g., run tests, query a database, perform SEO analysis) without embedding the knowledge in the model weights.

Pros: Modular and portable; encourages reuse and sharing of expertise; supports multiple agent runtimes.
Cons: Requires curation; not a transport protocol; adoption outside coding agents is limited.

Analogy: Comparable to installing a Python package or VS Code extension — an agent loads a skill to gain new abilities.

3 Inter-agent communication and coordination

3.1 Agent-to-Agent Protocol (A2A)

Publisher/Date: Google, April 2025.
Purpose: Provide a standard for multi-agent workflows where agents discover each other, authenticate and collaborate on tasks. A2A complements MCP by enabling peer-to-peer

delegation instead of just tool calls (source: https://a2a-protocol.org/).

How it works: The protocol uses HTTPS with JSON-RPC and Server-Sent Events (SSE) for streaming. Agents discover one another via **agent cards** (JSON metadata describing identity and capabilities) and then negotiate tasks. A2A emphasizes secure authentication, support for long-running tasks and modality-agnostic communication (text, audio, video). Communication flows through three phases: discovery (selecting remote agents), authorization (validating permissions) and communication (exchanging tasks and results).

Use cases: Distributed planning (e.g., one agent writes code while another tests it), cross-company collaboration, distributed reasoning.

Pros: Built on existing web standards (HTTP, SSE); enterprise-grade security; supports streaming and long-running interactions; encourages modular agent ecosystems.
Cons: Still in draft; depends on agent cards and infrastructure; requires registration and trust frameworks; not yet widely implemented outside Google partners.

Analogy: Similar to an email system combined with RPC — agents send structured messages or "requests" and receive streaming results, enabling back-and-forth collaboration.

3.2 Agent Connect Protocol (AConP)

Publisher/Date: AGNTCY project started from Outshift by Cisco (Internet of Agents white paper), updated July 2025.
Purpose: Provide a vendor-agnostic foundation for the emerging "Internet of Agents." Cisco's ACP layer serves as the "waist" of this network, supplying discovery, authentication and interaction protocols so that AI agents from any vendor or framework can find each other, verify capabilities and collaborate (source: https://agntcy.org/).

How it works: ACP is part of the broader Internet of Agents architecture, comprising several phases. In the *Discover* phase, agents describe their identity and skills in an Open Agent Schema Framework (OASF) record and register with a decentralized Agent Directory, allowing others to locate suitable capabilities (source: https://outshift-headless-cms-s3.s3.us-east-2.amazonaws.com/Internet_of_Agents_Whitepaper.pdf). In the *Compose* phase, agents invoke one another via the Agent Connect Protocol. This REST-based specification uses OpenAPI and JSON to define standard endpoints to run, interrupt or resume tasks and supports thread-based interactions. A distributed registry lets clients discover remote agents and even download and execute them locally. Additional components like Secure Low-latency Interactive Messaging (SLIM) handle stateful, multi-modal messaging, while observability and evaluation frameworks monitor

performance. Each agent advertises which invocation techniques (synchronous or asynchronous) it supports. However, an IETF draft notes that the current ACP specification only covers basic invocation and lacks built-in authentication, authorization and user confirmation (source: https://www.ietf.org/archive/id/draft-rosenberg-ai-protocols-00.html).

Use cases: Standardized invocation of specialized agents in complex workflows; cross-vendor orchestration of agents across networks; enabling developers to assemble multi-agent applications without bespoke integrations.

Pros: Creates a neutral foundation for agent discovery and interoperability; defines universal schemas and a directory for agent capabilities; supports both remote invocation and local distribution; integrates quantum-safe messaging and observability tools, encouraging an open ecosystem.
Cons: Still early and evolving; the current specification lacks robust lifecycle management and built-in authentication or user confirmation; adoption is limited to the AGNTCY collective; the full stack may be complex for small projects.

Analogy: Much like TCP/IP and HTTP provided a common "waist" for the traditional Internet, ACP supplies standardized plumbing that lets diverse AI agents speak the same language, connect through consistent endpoints, and form an interoperable agent ecosystem.

3.3 Agent Protocol (agentprotocol.ai)

Publisher/Date: AGI Inc., 2024.
Purpose: Offer a **universal REST API** for agent interactions so tools, agents and orchestrators can work together regardless of framework. It defines endpoints to **create tasks**, **execute steps**, **list tasks** and **retrieve artifacts** (source: https://agentprotocol.ai).

How it works: The Agent Protocol defines a minimal REST API. A client sends a goal to the agent via a POST /ap/v1/agent/tasks call to create a task, then drives the agent by issuing POST /ap/v1/agent/tasks/{task_id}/steps requests to carry out individual steps. The agent returns status updates and outputs for each step, and other endpoints allow listing tasks and downloading any artifacts. This step-based pattern abstracts away implementation details and makes it easy for any orchestrator to control an agent.

Use cases: Running AutoGPT-like agents programmatically; benchmarking and orchestrating multiple agent frameworks; building agent hubs.

Pros: Simple and language-agnostic; focuses on workflow management; widely adopted in open-source projects (AutoGPT, smol-developer).
Cons: Limited to step-based interactions (task/step model); doesn't address discovery or negotiation; separate from UI and payments.

Analogy: Think of this standard as a universal job ticketing system for AI agents. It lets you create a task, break it into steps, monitor progress and collect results through a single REST API. Just as a ticket in a help-desk system moves through various stages until it's resolved, the Agent Protocol's task-and-step model provides a clear workflow for agents.

3.4 Agent Interaction & Transaction Protocol (AITP)

Publisher/Date: aitp.dev (draft by community), 2025.
Purpose: Provide a secure, standardized way for an AI assistant to interact with service agents (e.g., an airline) across trust boundaries. The protocol defines chat threads and **capabilities** (payments, data requests, decisions) that agents can negotiate (source: https://aitp.dev).

How it works: Interactions are represented as threads similar to OpenAI's Threads API; each message can include structured data such as UI elements, payment requests or decisions. The specification emphasises security, auditability and extensibility.

Use cases: Booking travel via an airline's agent; negotiating service contracts; cross-organization decision making.

Pros: Designed for cross-trust interactions; leverages chat-based semantics; extensible.
Cons: Still early; specification may change; requires adoption by service providers.

Analogy: AITP functions like HTTP and HTML do for the web. Instead of users navigating websites, your assistant speaks directly to a business's agent using standardized threads and structured messages. It can exchange forms, payment requests and decisions across trust boundaries in the same way web browsers can visit any site via shared protocol

3.5 Agora Protocol

Publisher/Date: University of Oxford researchers, October 2024.
Purpose: Address the **Agent Communication Trilemma** (versatility, efficiency, portability) when coordinating networks of LLM agents (source: https://arxiv.org/abs/2410.11905) (source: https://agoraprotocol.org/).

How it works: Agora is a **meta-protocol** that leverages existing communication standards and chooses the right mode depending on frequency: standardized routines for common messages, natural language for rare messages and **LLM-generated code** for everything in between. This dual approach allows agents to adapt to changing interfaces and network members. The authors demonstrate emergent, self-organising behaviours in large networks.

Use cases: Research on large networks of specialized LLMs solving complex tasks; dynamic generation of communication routines.

Pros: Addresses scalability; flexible; emphasises emergent protocols; fully decentralised.
Cons: Research-oriented; lacks concrete implementation guidelines; depends on LLMs generating code at runtime.

Analogy: Similar to how humans converse: we use standard phrases for common tasks, full sentences for novel cases, and create ad-hoc jargon when needed.

3.6 Agent Network Protocol (ANP)

Publisher/Date: AgentNetworkProtocol.com consortium, 2025.
Purpose: Provide an **open, secure and efficient Internet of Agents** using a layered architecture (source: https://agentnetworkprotocol.com/specs/01-anp-technicalwhitepaper/) (source: https://agent-network-protocol.com/).

How it works: ANP comprises three layers:
1. **Identity & encrypted communication layer** – uses W3C Decentralised Identifiers (DID) and Elliptic Curve Diffie–Hellman (ECDHE) to establish peer-to-peer identities and encrypted channels, supporting multi-DID strategies and human authorization for high-risk operations.
2. **Meta-protocol layer** – negotiates which communication protocol to use through natural language proposals and AI-generated code; agents iterate until they agree on a protocol.
3. **Application protocol layer** – standardizes semantic descriptions of capabilities using JSON-LD and RDF; agents can share, evolve and reuse protocol definitions.

Use cases: Building a decentralized agent internet; cross-platform identity; dynamic protocol negotiation; regulated operations requiring user approval.

Pros: Strong identity and encryption; flexible meta-protocol negotiation; semantic web integration; aims for decentralized, self-governed network.
Cons: Complex; early-stage; adoption unknown; requires agents to implement DID and cryptography.

Analogy: Think of ANP as the **TCP/IP of agents**: the identity layer provides secure "IP addresses," the meta layer negotiates a "protocol," and the application layer defines the meaning of messages.

3.7 Summoner (SPLT Protocol) and decentralized agent networks

Publisher/Date: Summoner Network (SPLT), 2025.
Purpose: Build a decentralized infrastructure for cross-enterprise orchestration of autonomous agents. It emphasizes privacy, trust and state coordination across an open network (source: https://summoner.org/product).

How it works: The Summoner stack provides a Python/Rust SDK and high-performance relay servers. Agents create self-issued identities using Ed25519/X25519 key pairs, eliminating centralized certificate authorities. Trust is built through **behaviour-based reputation**: each agent maintains local scores for peers based on message timing, protocol compliance and reliability, with Merkle-tree proofs and decay mechanisms to prevent manipulation. Messages are end-to-end encrypted using Diffie–Hellman key exchange; the relay servers forward encrypted packets and enforce anti-replay protection. A two-phase discovery system moves new agents from a tentative pool to an active list based on engagement.

Use cases: Cross-enterprise agent networks, peer-to-peer marketplaces, privacy-focused collaborations.

Pros: Decentralized identity; strong encryption; behaviour-based trust; open-source; modular.
Cons: Complex reputation management; discovery may be slower than centralized registries; adoption still nascent.

Analogy: Similar to BitTorrent combined with PGP — each peer manages its own keys, reputations and connections without relying on a central authority.

4 Commerce and payment protocols

4.1 Agent Payments Protocol (AP2, also called A2P)

Publisher/Date: Google, April 2025.
Purpose: Provide a **secure, role-based payment standard** so agents can initiate transactions on behalf of users without exposing sensitive credentials (source: https://ap2-protocol.org/).

How it works: AP2 introduces three **cryptographically signed mandates**:
1. **Cart mandate** – user is present; it authorizes an agent to use a payment credential for a single cart (e.g., buying a pair of shoes).
2. **Intent mandate** – pre-approval for repeated purchases within certain limits (e.g., automatically restocking groceries).
3. **Payment mandate** – authorizes the payment provider to transfer funds from the user to the merchant.
The protocol involves a shopping agent, merchant endpoint, credential provider (issuing tokens), payment processor and network/issuer. It is payment-instrument agnostic and uses layered security to prevent collisions and fraud.

Use cases: Buying limited-edition items as soon as they drop; recurring subscriptions; restocking supplies.

Pros: Decouples user credentials from agents; supports various payment methods (cards, bank transfers, digital currencies); provides audit trails and cryptographic proof of user intent.
Cons: Complex to implement; requires cooperation among credential providers, merchants and networks; limited adoption outside Google's ecosystem.

Analogy: Similar to handing a signed check to a personal shopper — the shopper uses the check for a specific purchase, and the bank verifies signatures and amounts.

4.2 Agentic Commerce Protocol (ACP) – OpenAI/Stripe

Publisher/Date: OpenAI and Stripe, November 2025 (draft).
Purpose: Create an **open, programmatic checkout standard** that connects buyers, their agents and businesses for embedded commerce flows. It is separate from Google's AP2 and aims to work with any payment processor (source: https://agenticcommerce.dev).

How it works: ACP defines machine-readable schemas for product discovery, order creation, payment authorization and fulfillment. Businesses publish a **checkout configuration** via a REST API or MCP endpoint; agents call these endpoints to build carts,

apply discounts and submit orders. ACP supports complex flows (physical/digital goods, subscriptions, asynchronous purchases) and is **PCI compliant.** The specification includes OpenAPI and JSON Schema files and provides examples for integration (source: https://github.com/agentic-commerce-protocol/agentic-commerce-protocol).

Use cases: ChatGPT's instant checkout; e-commerce integration with AI assistants; marketplace transactions.

Pros: Open-source under Apache 2.0; designed for interoperability across agents and payment processors; businesses retain control over merchandising and fulfillment.
Cons: Draft status; reliant on adoption by merchants and PSPs; duplicates some functions of UCP and AP2.

Analogy: Comparable to the "Shopping cart API" for AI agents — a standard way to create carts, collect payment tokens and finalize orders with any merchant.

4.3 Universal Commerce Protocol (UCP)

Publisher/Date: Google with partners (Shopify, Etsy, Wayfair, Target, Walmart), announced January 2026.
Purpose: Collapse the **N×N integration problem** by providing a single open standard connecting consumer surfaces, businesses and payment providers. UCP unifies commerce from product discovery to payment, integrates with AP2 and A2A and is designed to work with existing retail infrastructure (source: https://ucp.dev/).

How it works: UCP specifies a **shared language** for describing commerce journeys, dynamic capability schemas and a modular architecture. It separates payment instruments from payment handlers and supports multiple transports (A2A, MCP, REST). Tokenized payments, verifiable credentials and dynamic capability discovery form the security core (source: https://developers.googleblog.com/under-the-hood-universal-commerce-protocol-ucp/).

Use cases: AI assistants that shop across multiple retailers; unified checkout surfaces across Google Search, Gemini, Android and partner platforms; cross-provider inventory integration.

Pros: Backed by major retailers; reduces complexity for AI platforms and merchants; extensible and security-first; interoperable with AP2 and A2A (source: https://blog.google/products/ads-commerce/agentic-commerce-ai-tools-protocol-retailers-platforms/).

Cons: New and potentially overlapping with ACP (OpenAI/Stripe); needs industry consensus; may favor Google's ecosystem.

Analogy: Similar to **HTTP** for e-commerce — one protocol that can handle all commerce interactions, regardless of the client or merchant.

4.4 Trusted Agent Protocol (TAP) – Visa

Publisher/Date: Visa (in collaboration with Cloudflare), October 2025.
Purpose: Provide a **cryptographic trust handshake** that allows merchants to differentiate trustworthy AI shopping agents from malicious bots (source: https://developer.visa.com/capabilities/trusted-agent-protocol/overview).

How it works: Visa runs an **Intelligent Commerce** program to vet and onboard AI agents; each trusted agent receives a unique cryptographic key. When an agent visits a merchant, it signs requests with its private key and attaches verifiable data: **agent intent** (why it's visiting), **consumer recognition** (token/loyalty ID or device identifier) and **payment information** (tokenized card or hashed credentials). Merchants verify the signature and decide whether to allow the agent. The system ensures signatures are time-bound and non-replayable, and merchants gain insight into the underlying consumer.

Use cases: Filtering out malicious bots in agentic commerce; improving fraud prevention; enabling merchants to personalize interactions.

Pros: Enhances trust and safety; leverages Visa's global network; allows merchants to maintain existing bot protections while welcoming approved agents.
Cons: Requires registration and approval by Visa; centralizes trust; may limit open participation; early stage.

Analogy: Functions like a **VIP list** at a club — merchants only admit agents with a verifiable, cryptographically signed pass.

4.5 Agent Pay – Mastercard

Publisher/Date: Mastercard, mid-2025.
Purpose: Provide infrastructure and standards to support secure, scalable and trusted payments in agentic commerce (source: https://www.mastercard.com/us/en/business/artificial-intelligence/mastercard-agent-pay.html).

How it works: Mastercard's Agent Pay program leverages network tokens, biometric authentication and a universal data exchange protocol to ensure registered agents can transact. Only registered agents can process payments; order intent is verified and consent is captured; consumers maintain control through passkeys and tokenization. While details are marketing-heavy, the program emphasises trust, security and global scalability.

Use cases: Enabling agentic payments across Mastercard's network; cooperating with UCP and other standards.

Pros: Backed by a major card network; integrated with existing payment rails; focus on consumer consent and identity.
Cons: Limited technical details publicly available; may compete with Visa's TAP and Google's AP2; potential for fragmentation.

Analogy: Like handing your assistant a single-use digital card and a hall pass — Mastercard issues a unique token to a registered agent so it can only buy what you've authorized, and each purchase still requires your biometric consent.

4.6 x402 Protocol (micropayments for agents)

Publisher/Date: x402 working group (2025).
Purpose: Revive the **HTTP 402 "Payment Required"** status code to enable instant, low-value micropayments between web services and AI agents (source: https://www.x402.org/).

How it works: When a client (e.g., an agent) requests data from a server and payment is required, the server replies with a 402 response including the amount and recipient address. The client crafts a signed payment payload (often using stablecoins), resends the request with the payment in an HTTP header, and the server verifies via a blockchain facilitator. The flow is stateless, fits existing HTTP stacks and supports per-use pricing. x402's design is developer-friendly (middleware can enforce payments) and targets micropayments where fees must be negligible.

Use cases: Pay-per-use API calls, machine-to-machine data access, anti-spam micro-fees.

Pros: Works with existing web infrastructure; enables dynamic micro-economies; fosters machine-to-machine commerce.
Cons: Depends on blockchain settlement; introduces payment friction; requires robust authorization controls.

Analogy: Comparable to turning HTTP into a vending machine — if a resource costs one cent, the server returns a 402, the agent pays, and the server responds with the data.

5 User interaction and UI protocols

5.1 Agent-User Interaction Protocol (AG-UI)

Publisher/Date: CopilotKit (OSS LangGraph/CrewAI), 2025.
Purpose: Establish a **bi-directional event protocol** connecting agent back-ends with user interfaces. It standardizes how agent state, UI intents and user interactions are streamed. AG-UI is distinct from UI description formats (A2UI, Open-JSON-UI) — it is the **transport layer** rather than the language of UI (source: https://docs.ag-ui.com).

How it works: AG-UI uses streaming channels (SSE, WebSockets, HTTP streams) to exchange events such as `RUN_STARTED`, `STATE_DELTA`, `TOOL_CALL_START/END` and `USER_ACTION`. The protocol supports shared state, tool output streaming, thinking steps and interrupts for human-in-the-loop sessions. It also allows generative UI specs (e.g., A2UI or Open-JSON-UI) to be embedded within state deltas.

Use cases: Real-time interactive applications (e.g., chat with AI that shows progress bars, charts and tool results); streaming reasoning traces.
Pros: Enables incremental UI updates; supports multimodality (attachments, media); decouples UI description from transport.
Cons: Requires integration with front-end frameworks; still evolving; not widely standardized across ecosystems.

Analogy: Like **WebSockets plus Redux** — a real-time event bus that synchronizes agent state and user actions.

5.2 Agent to UI Protocol (A2UI) and Open-JSON-UI

Publisher/Date: Google, December 2025.
Purpose: Provide a **declarative, streaming-friendly UI specification** so agents can describe interactive interfaces that render natively on web, mobile and desktop without executing arbitrary code. A2UI was designed alongside AG-UI to solve the UI generation problem across trust boundaries (source: https://a2ui.org).

How it works: Agents generate flat, streaming JSON describing surfaces, components and data models. A2UI prohibits arbitrary JavaScript and uses pre-approved components to avoid code injection. It supports progressive rendering (UI can be updated as new JSON

fragments arrive) and is framework-agnostic (renderers exist for Angular, React, Flutter, etc.) (source: https://developers.googleblog.com/introducing-a2ui-an-open-project-for-agent-driven-interfaces/).

Use cases: AI assistants that produce interactive dashboards, forms or charts; generative business applications; UIs embedded within chat surfaces.

Pros: Secure by design; incremental streaming; can be rendered in any framework; works with A2A and UCP.
Cons: New specification; limited component library; still lacks community-agreed schema for complex layouts.

Analogy: A2UI is like **HTML/JSX** for AI — a declarative markup for building UI, but delivered as streaming JSON rather than text markup.

6 File-based guidance and skills

6.1 Agentic AI Foundation's AGENTS.md and Anthropic's CLAUDE.md

Purpose: Provide reproducible, context-rich instructions for coding agents.

AGENTS.md: This markdown file lives in a repository and acts as a **README specifically for AI coding agents**. It contains build steps, testing instructions, conventions and any other guidance needed for automated code generation. It supplements but does not replace `README.md` (source: https://agents.md). Over 60k open-source projects have adopted it.

CLAUDE.md: Anthropic's Claude Code automatically loads a `CLAUDE.md` file when starting a coding session. The file documents bash commands, core files, code style guidelines and developer environment setup, allowing developers to tune the agent's behavior (source: https://www.anthropic.com/engineering/claude-code-best-practices).

Use cases: Guiding LLM coding agents to build, test and deploy software; customizing an agent's environment.

Pros: Simple to adopt; improves reproducibility and reduces hallucination; integrated into major LLM products.
Cons: Not a communication protocol; limited to coding; reliant on file discipline.

Analogy: AGENTS.md/CLAUDE.md are like **checklists or runbooks** for AI developers, telling the agent exactly how to operate within a repository.

6.2 llms.txt and Agents.json

Publisher/Date: Jeremy Howard proposed **llms.txt** (September 2024); Wildcard introduced **agents.json** (2025).
Purpose: Provide web-accessible files that optimize websites and APIs for LLM consumption.

llms.txt: A simple text file at a site's root that includes an H1 with the site name, a succinct summary and lists of links to Markdown versions of pages. It helps LLMs quickly understand a site's context without scraping HTML, coexisting with robots.txt and sitemaps (source: https://llmstxt.org).

agents.json: An OpenAPI-compatible JSON specification that formalizes **contracts** for AI agents interacting with APIs and websites. It optimizes the schema for LLMs, enforces stateless orchestration and requires minimal changes to existing APIs (source: https://github.com/wild-card-ai/agents-json).

Use cases: Websites providing curated information for LLMs; API providers exposing structured documentation; improving agent reliability when using external services.

Pros: Human- and machine-readable; reduces context length; encourages best practices; open source.
Cons: Adoption still limited; site owners must maintain new files.

Analogy: Similar to **robots.txt** but for AI, telling LLMs "Here's what you need to know about this site."

7 Identity, networks and operating systems

7.1 Agentic JWT (A-JWT)

Publisher/Date: University of Chicago (Abhishek Goswami) / Internet Engineering Task Force draft, 2025.
Purpose: Provide a secure delegation and authorization protocol for agentic AI systems by extending JSON Web Tokens (JWTs) with agent identity, user intent and workflow context (source: https://arxiv.org/abs/2509.13597).

How it works: The design introduces a **dual-faceted token**: an **intent token** that binds each agent action to a cryptographically verifiable user intent and optional workflow step, and a standard **access token** for deterministic components. The intent token carries new claims containing the agent's identity, represented by an `agent_checksum` (a SHA-256 hash of the agent's prompt, tool definitions and configuration), and a chained delegation assertion recording which downstream agent may perform the task (source: https://datatracker.ietf.org/doc/draft-goswami-agentic-jwt/). A-JWT defines a new OAuth 2.0 authorization grant type (**agent_checksum**) that allows authorization servers to issue tokens based on the agent's runtime identity instead of static credentials. A lightweight client-side **shim library** computes the agent checksum at runtime, verifies the agent's registration, tracks workflow state, mints intent tokens and derives per-agent proof-of-possession keys. The protocol remains backward-compatible with existing OAuth 2.0 flows but adds agent-aware claims and proofs.

Use cases: Securing API calls by autonomous agents; enforcing separation between user intent and agent execution; preventing replay, impersonation and prompt-injection attacks; providing auditable trails for multi-agent workflows.

Pros: Cryptographically binds agent actions to user intent; enables verifiable delegation and fine-grained workflow control; integrates with established OAuth/JWT infrastructure; supports zero-trust architectures with sub-millisecond overhead.
Cons: Still a draft specification and subject to change; requires new identity-provider components and agent registration processes; adds complexity and potential latency to agent workflows; adoption and standardization remain uncertain.

Analogy: A-JWT is like a **power-of-attorney** document for every API call – each token carries a signed statement of the user's intent, the agent's identity and the workflow step, ensuring that an agent cannot act beyond its delegated authority.

7.2 Language Model Operating System (LMOS)

Publisher/Date: Eclipse Foundation (project led by Red Hat, IBM and others), 2025.
Purpose: Provide a **platform-agnostic operating system** for building, orchestrating and scaling AI agents across cloud and enterprise environments. LMOS includes an **Agent Definition Language (ADL)** that allows domain experts to define agent behaviour in a structured, model-neutral manner (source: https://eclipse.dev/lmos/).

How it works: LMOS runs on Kubernetes and integrates with enterprise infrastructure like

Istio and the JVM. Its agent framework (ARC) and platform manage agent life cycles, discovery, routing and observability. ADL allows non-programmers to specify tasks, resources and policies, which LMOS interprets to orchestrate underlying models and services.

Use cases: Large enterprises deploying fleets of AI agents; orchestrating multi-model workflows; bridging domain experts and engineers.

Pros: Open platform; leverages mature cloud tooling; provides structured language for agent behaviour; encourages standardization across vendors.
Cons: Heavyweight; requires Kubernetes expertise; still early stage.

Analogy: LMOS plays the role of a **cloud OS for agents**, akin to how Android OS manages apps on a phone.

8 Summary table

The table below summarises the most important protocols with their primary domain and a succinct description. “Launch Year” refers to when the protocol became public or widely discussed.

Protocol	Domain /Layer	Launch Year	Key Purpose & Example Use
MCP (Model Context Protocol)	Tool integration	2024	Standardizes how agents discover and call external tools, injecting structured context into prompts (source: https://modelcontextprotocol.io/).
A2A (Agent-to-Agent Protocol)	Inter-agent comms	2025	Enables agents to discover, authenticate and collaborate via JSON-RPC and SSE; supports streaming and long-running tasks (source: https://a2a-protocol.org/).
AP2 / A2P (Agent Payments Protocol)	Payments	2025	Defines cryptographically signed cart, intent and payment mandates for agents to perform purchases securely (source: https://ap2-protocol.org/).
ACP – Agentic Commerce Protocol (OpenAI/Stripe)	Commerce	2025	Provides an open standard for programmatic checkout; supports complex flows and integrates with any merchant or PSP (source: https://agenticcommerce.dev).
UCP (Universal Commerce Protocol)	Commerce	2026	Aims to be the universal agentic commerce layer; unifies discovery, purchase and payment across consumer surfaces and retailers (source: https://ucp.dev).
A2UI	UI description	2025	Declarative JSON format for streaming agent-generated UIs; secure by design and framework-agnostic (source: https://a2ui.org).
AG-UI	UI transport	2025	Event-driven protocol connecting agents and UIs; streams state deltas, tool calls and user actions (source: https://docs.ag-ui.com).
AGENTS.md / CLAUDE.md	File format	2024–25	Markdown files providing build steps and instructions for coding agents (source: https://agents.md) (source: https://www.anthropic.com/engineering/claude-code-best-practices).
llms.txt / agents.json	File/API specs	2024–25	llms.txt summarises websites for LLMs (source: https://llmstxt.org); agents.json adapts OpenAPI for LLM-friendly API contracts (source: https://github.com/wild-card-ai/agents-json).
Agent Protocol (agentprotocol.ai)	Inter-agent API	2024	Defines REST endpoints to create tasks, execute steps and retrieve artifacts (source: https://agentprotocol.ai).

Protocol	Domain /Layer	Launch Year	Key Purpose & Example Use
Agent Connect Protocol (AConP)	Invocation & orchestration	2025	Offers a REST/OpenAPI API to run, interrupt and resume agents across frameworks, with a registry for discovering and even downloading them (source: https://agntcy.org/)
AITP (Agent Interaction & Transaction Protocol)	Secure multi-agent	2025	Draft spec enabling secure agent-to-agent threads with capabilities (payments, data requests, decisions) (source: https://aitp.dev).
Agora Protocol	Meta-protocol	2024	Uses structured routines for frequent messages, natural language for rare ones and LLM-generated code for intermediate cases to solve the communication trilemma (source: https://agoraprotocol.org/).
ANP (Agent Network Protocol)	Identity & network	2025	Three-layer protocol providing DIDs for identity, meta-protocol negotiation and semantic application descriptions (source: https://agent-network-protocol.com/)
Summoner / SPLT	Decentralized network	2025	Decentralized infrastructure with self-issued identities, behavior-based reputation and end-to-end encrypted relay routing (source: https://summoner.org/product).
x402	Micropayments	2025	Utilizes HTTP 402 status to enable micropayments using signed blockchain transactions (source: https://www.x402.org/).
Visa Trusted Agent Protocol (TAP)	Commerce trust	2025	Cryptographic handshake verifying AI shopping agents; includes agent intent, consumer recognition and payment info (source: https://developer.visa.com/capabilities/trusted-agent-protocol/overview).
Mastercard Agent Pay	Commerce payments	2025	Program for secure agentic payments using network tokens and verified order intent (source: https://www.mastercard.com/us/en/business/artificial-intelligence/mastercard-agent-pay.html).
Agent Skills / SKILL.md	Skills packaging	2024	Packages procedural knowledge and instructions for agents to load on demand (source: https://agentskills.io).
LMOS (Language Model Operating System)	Agent OS	2025	Cloud-native platform with Agent Definition Language for specifying agent behaviour (source: https://eclipse.dev/lmos/).
A-JWT (Agentic JWT)	Authorization & delegation	2025	Extends JWT to cryptographically bind each agent action to user intent and workflow using an agent checksum and intent token (source: https://datatracker.ietf.org/doc/draft-goswami-agentic-jwt/).

9 Protocol x Capabilities x Maturity

The proliferation of protocols highlights competing visions. Google champions A2A, AP2 and UCP, while Anthropic promotes MCP and skills, OpenAI collaborates with Stripe on ACP and uses AG-UI and Open-JSON-UI. Card networks (Visa, Mastercard) are building trust layers and payment rails. Decentralized projects like Summoner and ANP aim for peer-to-peer autonomy.

The result is a complex landscape where interoperability is both essential and challenging.

Legend

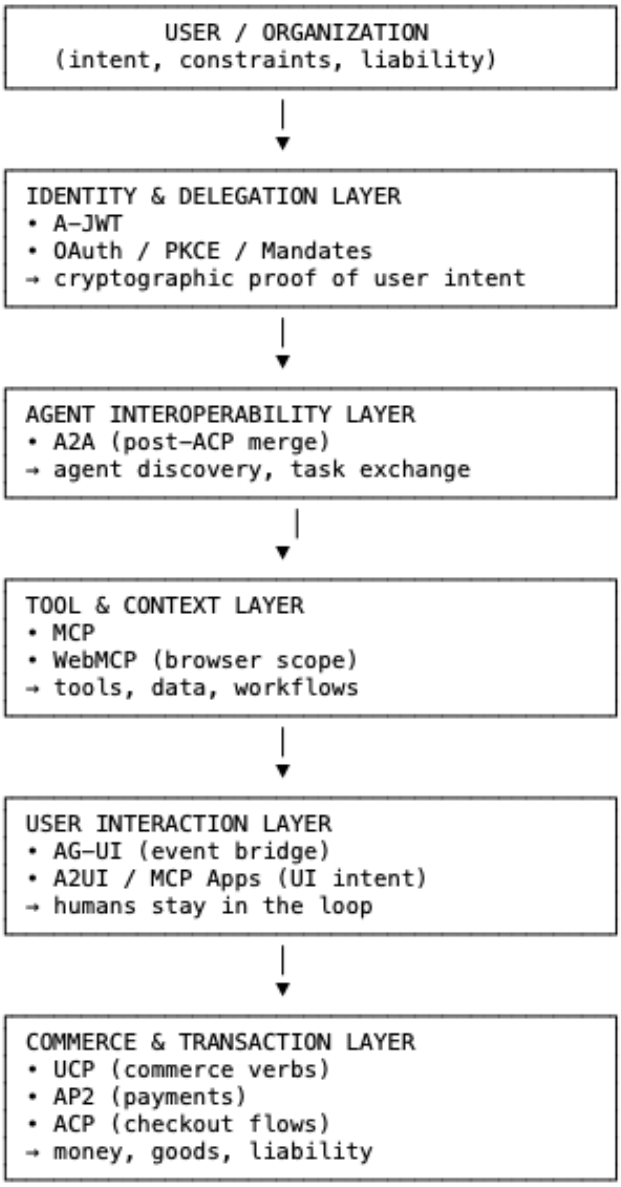
- ✅ = native / core
- ◐ = partial / indirect
- ⭕ = not in scope

Protocol	Identity / Auth	Agent Interoperability	Tools / Context	UI / UX	Commerce / Payments	Governance / Origin	Maturity (2026 view)
A-JWT	✅ core (intent-bound delegation)	◐ (trust layer)	◐	⭕	◐ (payment authorization)	IETF draft	**Early / foundational**
A2A	◐	✅ core	◐	⭕	◐ (transport for AP2/UCP)	Linux Foundation	**Emerging → consolidating**
AConP (Agent Connect Protocol)	◐ (limited today)	✅ core (directory + invocation + messaging stack)	◐	⭕	⭕	AGNTCY / Cisco Outshift	**Early / infrastructural**
SPLT (Summoner Protocol)	✅ (self-issued keys + encryption)	✅ core (decentralized agent networking + relays)	⭕	◐	⭕	Summoner Network	**Early / nascent adoption**
Agent Protocol	◐	◐	◐	⭕	⭕	Community (research-driven)	**Early / conceptual**
ANP (Agent Network Protocol)	◐	✅ (network-level routing)	⭕	⭕	⭕	Academic / OSS	**Early / experimental**
Agora Protocol	◐	◐ (market-style coordination)	⭕	⭕	◐	Research-led OSS	**Experimental**
AITP (AI Task Protocol)	⭕	◐ (task exchange patterns)	◐	⭕	⭕	Community	**Early / niche**
MCP	◐ (scoped access)	⭕	✅ core	◐ (via extensions)	⭕	AAIF	**Maturing / widely adopted**
WebMCP	◐	⭕	✅	◐	⭕	W3C incubation	**Experimental**
AG-UI	⭕	⭕	◐	✅ core	⭕	OSS	**Early but practical**
A2UI	⭕	⭕	⭕	✅ core	⭕	Google-led OSS	**Early / fast-moving**
MCP Apps (ext)	◐	⭕	◐	✅	⭕	MCP Working Group	**Draft / likely standard**
UCP	◐ (identity linking)	◐	◐	◐	✅ core	OSS consortium	**Very new / strategic**
AP2 (A2P)	◐ (mandates, consent)	◐	⭕	⭕	✅ core	Google-led OSS	**Early / high-stakes**
TAP (Trusted Agent Protocol – Visa)	✅ core (agent authenticity + authorization proof)	◐ (trust handshake, not orchestration)	⭕	⭕	✅ core	Visa	**Early / commercial-critical**
Agent Pay	◐	◐	⭕	⭕	✅ core	Payments ecosystem	**Early / commercial**
x402	◐ (HTTP auth extension)	⭕	⭕	⭕	◐	Community / payments	**Experimental**
ACP (OpenAI + Stripe)	◐	⭕	⭕	◐	✅ core	OSS draft	**Draft / ecosystem-specific**
LMOS	⭕	⭕	◐ (OS-level abstractions)	◐	⭕	Community / research	**Conceptual / early**
SKILL.md	⭕	⭕	◐ (capability declaration)	⭕	⭕	Community	**Emerging / lightweight**
AGENTS.md	⭕	⭕	◐ (instruction context)	⭕	⭕	AAIF	**Adopted / stable**
llms.txt	⭕	⭕	◐ (content discovery)	⭕	⭕	Community	**Adopted / lightweight**
agents.txt	⭕	◐ (basic discovery)	⭕	⭕	⭕	Community	**Speculative**

10 Canonical Agentic Stack Diagram

As agentic AI systems mature, a clear architectural pattern is emerging. Despite a fragmented protocol landscape, most credible initiatives converge on a **layered stack** that separates concerns between trust, coordination, execution, interaction, and monetization.

This diagram expresses our canonical view of that stack:
Identity → Agent Interoperability → Tools & Context → User Interaction → Commerce.



11 Key takeaways and future outlook

- **Explosion of standards:** From 2024 onwards the agentic landscape saw an explosion of protocols, each solving a different layer (tools, communication, payments, UI, identity). This fragmentation reflects rapid innovation but also creates confusion and interoperability challenges. Stakeholders should evaluate which layer they need and look for multi-protocol compatibility (e.g., A2A+AP2+UCP or MCP+ACP).

- **Shift toward open, REST-based designs:** Most protocols leverage established web standards (HTTP, SSE, JSON-RPC, REST) rather than invent new transports. This increases adoptability and makes integration easier. Decentralized efforts (ANP, Summoner) still rely on cryptography and peer-to-peer messaging but aim to remain compatible with web infrastructure.

- **Payments as a battleground:** Google's AP2, OpenAI/Stripe's ACP, Visa's TAP, Mastercard's Agent Pay and the x402 micropayment standard all address agent-led commerce from different angles. Expect convergence or bridging frameworks in future to avoid a fractured market.

- **UI separation of concerns:** AG-UI emphasises transport and lifecycle events, while A2UI and Open-JSON-UI handle UI description. Recognising this distinction helps developers choose the right protocols for their needs. Future UI standards may unify these efforts.

- **Identity & trust:** Decentralized identity (ANP, Summoner) and trust frameworks (Visa TAP, MasterCard Agent Pay) highlight the importance of verifiable identity and reputation in agentic ecosystems. Collaboration between these approaches is vital to prevent walled gardens.

- **Secure delegation and identity tokens:** The emergence of Agentic JWT highlights the need for cryptographically verifiable agent identity and intent binding. By extending OAuth and JWT with agent checksums and intent tokens, A-JWT brings zero-trust semantics into agentic AI systems.

- **Future trends:** We expect consolidation around a few dominant protocols (likely MCP for tools, A2A for inter-agent communication and one of the commerce standards), alongside niche protocols for specialized domains (e.g., micropayments). Emerging research (Agora) may inspire adaptive meta-protocols that let agents negotiate their own communication schemes.

The future agent stack will be A-JWT + A2A + MCP at the core,
with AG-UI/A2UI for humans and UCP/AP2 where money changes hands.