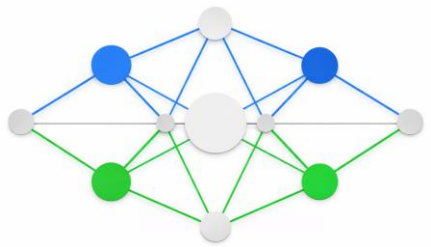


OCP

OpenCognition Protocol

"Where Minds Meet Machines"

OpenCognition Protocol (OCP)

Python SDK Technical Specification v1.0

Introduction	November 2025
License	MIT Licenses
Conformance Identifier	OCP-SDK-PYTHON-SPEC-1.0
Specification	docs.opencognitionprotocol.org
Maintainer	OCP Technical Steering Committee

1. Overview

1.1 Purpose

This document is the authoritative technical specification for the OCP Python SDK (`ocp-protocol`). It defines the public API surface, class contracts, method signatures, behavioral requirements, error semantics, concurrency model, and conformance criteria for the reference Python implementation of the OpenCognition Protocol.

All public classes, methods, and functions described in this document constitute the SDK's stable API. Implementations MUST conform to the contracts specified here. Internal (underscore-prefixed) members are not part of the public API and MAY change without notice.

1.2 Scope

The SDK implements all five layers of the OCP protocol stack:

Layer	SDK Module(s)	Section
Layer 1: Transport & Discovery	`transport`, `registry`	§8, §9
Layer 2: Trust & Identity	`identity`, `trust`, `crypto`	§4, §5, §6
Layer 3: Knowledge Exchange	`knowledge`, `pvl`	§7, §10
Layer 4: Collaboration	`consensus`, `messages`	§11, §12
Layer 5: Application	`agent`	§3
Cross-cutting: Key Recovery	`recovery`	§13

1.3 Conventions

- Method signatures use Python type hint syntax.
- `async` methods MUST be awaited. Synchronous wrappers are not provided.
- All timestamps are UTC `datetime` objects or ISO 8601 strings ending in `Z`.
- All binary data is represented as `bytes` internally and base64url-encoded when serialized to JSON.
- The keywords MUST, MUST NOT, SHOULD, SHOULD NOT, and MAY are per RFC 2119.

1.4 Installation

```
pip install ocp-protocol        # Core SDK
pip install ocp-protocol[dev]   # Development dependencies
```

2. Module Architecture

```
ocp/
├── __init__.py          Public re-exports (§2.1)
├── constants.py         Protocol constants (§2.2)
├── exceptions.py        Exception hierarchy (§2.3)
├── crypto.py            Cryptographic primitives (§4)
├── identity.py          Agent identity & DID documents (§5)
├── messages.py          OCPUMF builder & validator (§6)
├── knowledge.py         Knowledge types (§7)
├── transport.py         WebSocket & HTTP transports (§8)
├── registry.py          Agent Registry client (§9)
├── pvl.py               Privacy Validation Layer (§10)
├── consensus.py         Consensus protocol (§11)
├── trust.py             Trust, vouching, bonds (§12)
├── recovery.py          Shamir SSS key recovery (§13)
├── agent.py             High-level Agent class (§3)
├── cli.py               CLI tools (§14)
└── compliance.py        Compliance checker (§15)
```

2.1 Public Exports

The top-level `ocp` package re-exports all public symbols. Users SHOULD import from `ocp` directly:

```
from ocp import Agent, MessageType, InsightPackage, Bond
```

Module-level imports are also supported for targeted use:

```
from ocp.crypto import SigningKeyPair, sha3_256
from ocp.pvl import enforce_pvl
```

The `__all__` list in `ocp/__init__.py` is the definitive enumeration of public symbols. Any symbol not in `__all__` is internal.

2.2 Constants Module (`ocp.constants`)

All protocol-level constants are centralized in `ocp.constants`. SDK code MUST reference these constants rather than using magic numbers. The constants are normative — changing any value produces a non-conforming implementation.

Constant	Type	Value	Specification Reference
`OCP_VERSION`	`str`	`"1.0"`	§7.1
`MAX_MESSAGE_SIZE`	`int`	`16_777_216`	§9
`MAX_PAYLOAD_SIZE`	`int`	`10_485_760`	§9
`DEFAULT_TTL`	`int`	`3600`	§4.2
`MAX_TTL`	`int`	`86_400`	§4.2
`AUTH_HANDSHAKE_TIMEOUT`	`float`	`5.0`	§3.1.1
`ACK_TIMEOUT`	`float`	`30.0`	§3.4
`TASK_DEFAULT_TIMEOUT`	`int`	`300`	§6.1
`MAX_RETRY_COUNT`	`int`	`5`	§3.4

RETRY_BASE_DELAY	float	1.0	§3.4
REGISTRY_RECORD_DEFAULT_TTL	int	86_400	§3.2.2
MAX_VOUCH_DURATION_SECONDS	int	31_536_000	§4.2.3
MAX_BOND_DURATION_SECONDS	int	31_536_000	§4.3.2
KEY_ROTATION_RECOMMENDED_SECONDS	int	7_776_000	§8.2
TRUST_SCORE_MIN	float	0.0	§4.2.2
TRUST_SCORE_MAX	float	1.0	§4.2.2
MAX_EPSILON	float	10.0	§5.1.3
DEFAULT_RECOVERY_THRESHOLD	int	3	§8.3.2
DEFAULT_RECOVERY_SHARES	int	5	§8.3.2
MAX_RECOVERY_SHARES	int	255	§8.3.2
HKDF_INFO_AES_KEY	bytes	b"ocp-v1-aes-key"	§7.3
HKDF_INFO_RECOVERY	bytes	b"ocp-v1-recovery-share"	§8.3.3
WS_SUBPROTOCOL	str	"ocp.v1"	§3.1.1

2.3 Exception Hierarchy

```
OCPError
├── OCPAuthError              OCP-401, OCP-403
├── OCPTransportError         OCP-502, OCP-503
├── OCPValidationError        OCP-400
├── OCPTrustError             OCP-403
├── OCPPrivacyViolation       PVL-001 through PVL-006
├── OCPTimeoutError           OCP-408
├── OCPRateLimitError         OCP-429
├── OCPAgentNotFoundError     OCP-404
├── OCPConsensusError         Consensus-specific failures
└── OCPRecoveryError          Key recovery failures
```

Contracts:

- All exceptions inherit from `OCPError`.
- `OCPError` exposes a `code: str | None` attribute containing the OCP error code.
- `OCPPrivacyViolation` exposes an additional `pvl_code: str` attribute.
- `OCPRateLimitError` exposes a `retry_after: int` attribute (seconds).
- All exceptions MUST have a meaningful `__repr__` that includes the error code.

- SDK methods MUST raise the most specific applicable exception.
- SDK methods MUST NOT raise exceptions outside this hierarchy for protocol-level failures. Python built-in exceptions (e.g., `TypeError`, `ValueError`) are permitted for programming errors.

3. Agent Class (`ocp.agent.Agent`)

The `Agent` class is the primary entry point for the SDK. It composes identity, transport, registry, trust, knowledge, consensus, and recovery into a single coherent interface.

3.1 Construction

```python
@dataclass
class Agent:
    name: str
    capabilities: list[str]
    domains: list[str]
    network: str = "mainnet"
    service_endpoint: str = ""
    registry_url: str = "https://registry.ocp.foundation/ocp/v1"
```

Post-initialization behavior:

On `__post_init__`, the Agent MUST:

1. Generate a fresh `AgentIdentity` (Ed25519 + X25519 keypairs, DID derivation).
2. Initialize a `RegistryClient` pointed at `registry_url`.
3. Initialize a `RecoveryManager` bound to the new identity.
4. Initialize empty bond, vouch, and consensus stores.

The Agent MUST NOT make any network calls during construction.

3.2 Properties

Property	Type	Description
`agent_id`	`str`	The agent's DID. Immutable after construction.
`identity`	`AgentIdentity`	Full cryptographic identity.
`trust_level`	`TrustLevel`	Current trust level derived from local state.
`bonds`	`dict[str, Bond]`	Active (non-expired, non-revoked) bonds keyed by peer DID.
`recovery_manager`	`RecoveryManager`	Key recovery manager.

3.3 Lifecycle Methods

Method	Signature	Description
`register`	`async () -> dict[str, Any]`	Register on the OCP network via the Agent Registry.
`connect_ws`	`async (url: str | None) -> None`	Establish WebSocket transport.
`connect_http`	`async (url: str | None) -> None`	Configure HTTP transport.
`close`	`async () -> None`	Close all transports and the registry client.

`register()` behavior:

1. Build an `AgentRecord` from the agent's name, capabilities, domains, and endpoints.
2. Sign the registration payload with the agent's signing key.
3. POST to the registry's agent registration endpoint.
4. Return the registry response dict.
5. Raise `OCPTransportError` on network failure.

`close()` behavior:

1. Close WebSocket transport if connected.

2. Close HTTP transport if configured.
3. Close registry client.
4. The method MUST be idempotent — calling it multiple times MUST NOT raise.

3.4 Discovery

```python
async def discover(
    self,
    domain: str | None = None,
    capability: str | None = None,
    min_trust_level: int = 0,
    limit: int = 20,
) -> list[dict[str, Any]]
```

Behavior:

- Delegates to `RegistryClient.discover()`.
- Returns a list of agent record dicts matching the filters.
- Returns an empty list if no matches are found (MUST NOT raise).
- Raises `OCPTransportError` on network failure.

3.5 Messaging

```python
async def send_message(
    self,
    to: str,
    msg_type: MessageType,
    payload: dict[str, Any],
    priority: Priority = Priority.NORMAL,
    require_ack: bool = False,
) -> dict[str, Any]
```

Behavior:

1. Build an OCPUMF message via `MessageBuilder`.
2. Sign the message with the agent's signing key.

3. Send via the first available transport (HTTP preferred, WebSocket fallback).
4. Return the transport response dict.
5. Raise `OCPValidationError` if no transport is configured.

```python
async def broadcast(
    self,
    payload: dict[str, Any],
    tags: list[str] | None = None,
) -> dict[str, Any]
```

Behavior:

- Set receiver to the broadcast address (`did:ocp:<network>:broadcast`).
- Set `broadcast: true` on the receiver field.
- Attach tags to metadata.

3.6 Knowledge Sharing

```python
async def share(
    self,
    knowledge: InsightPackage | EmbeddingPackage | ModelDelta,
    to: str,
) -> dict[str, Any]
```

Behavior:

1. Call `knowledge.to_payload()` to serialize.
2. Call `enforce_pvl(payload)` — raises `OCPPrivacyViolation` on failure.
3. If a bond exists with the receiver, verify the bond permits the knowledge type — raise `OCPTrustError` if not.
4. Send as a `knowledge_share` message.

Invariant: No knowledge payload MUST ever leave the agent without passing PVL. This is enforced at the SDK level, not delegated to the transport or node.

3.7 Bonding

Method	Signature	Description
`request_bond`	`async (peer_id, permissions?, duration_days?) -> dict`	Send bond request to peer.
`accept_bond`	`(peer_id, bond) -> None`	Record accepted bond locally.
`revoke_bond`	`(peer_id) -> Bond | None`	Revoke a bond. Returns the revoked bond or `None`.

3.8 Vouching

Method	Signature	Description
`vouch_for`	`(subject_id, domains) -> Vouch`	Create and sign a vouch.
`receive_vouch`	`(vouch) -> None`	Record a received vouch.

3.9 Task Delegation

```python
async def delegate_task(
    self,
    to: str,
    task_type: str,
    description: str,
    input_data: dict[str, Any],
    max_duration: int = 300,
    required_capabilities: list[str] | None = None,
) -> dict[str, Any]
```

Behavior:

1. Check bond permits task delegation — raise `OCPTrustError` if not.
2. Build a `task_request` payload with auto-generated `task_id`.
3. Send to the receiver.
4. Return the transport response.

3.10 Consensus

```python
def initiate_consensus(self, config: ConsensusConfig) -> ConsensusRound
```

Behavior:

- Create a `ConsensusRound` with a generated `consensus_id`.
- Store the round in `_active_consensus`.
- Return the round (caller broadcasts the initiation payload separately).

4. Cryptographic Primitives (`ocp.crypto`)

This module provides all cryptographic operations required by the protocol. It MUST NOT implement cryptographic algorithms directly — it MUST delegate to the `cryptography` library (≥ 43.0) and `pynacl` (≥ 1.5.0).

4.1 Encoding Functions

Function	Signature	Description
`b64url_encode`	`(data: bytes) -> str`	Base64url without padding (RFC 4648 §5).
`b64url_decode`	`(s: str) -> bytes`	Decode base64url, handling missing padding.

Contracts:

- `b64url_decode(b64url_encode(x)) == x` for all `x: bytes`.
- Output strings MUST contain only `[A-Za-z0-9_-]`.
- No `=` padding characters in output.

4.2 Hashing

Function	Signature	Description
`sha3_256`	`(data: bytes) -> bytes`	SHA-3-256 digest (32 bytes).
`sha3_256_hex`	`(data: bytes) -> str`	SHA-3-256 as 64-char lowercase hex.

Contracts:

- MUST use FIPS 202 SHA-3 (Keccak), not SHA-256.
- `len(sha3_256(x)) == 32` for all inputs.
- `len(sha3_256_hex(x)) == 64` for all inputs.
- Deterministic: `sha3_256(x) == sha3_256(x)` always.

4.3 Ed25519 Signing (`SigningKeyPair`)

```python
@dataclass(frozen=True)
class SigningKeyPair:
    private_key: Ed25519PrivateKey
    public_key: Ed25519PublicKey
```

Method/Property	Signature	Description
`generate`	`classmethod () -> Self`	Generate from OS CSPRNG.
`from_private_bytes`	`classmethod (data: bytes) -> Self`	Restore from 32-byte private key.
`sign`	`(data: bytes) -> bytes`	Ed25519 signature (64 bytes).
`verify`	`(signature: bytes, data: bytes) -> None`	Raises InvalidSignature on failure.
`private_key_bytes`	`-> bytes`	Raw 32-byte private key.
`public_key_bytes`	`-> bytes`	Raw 32-byte public key.
`public_key_b64url`	`-> str`	Base64url-encoded public key.

`public_key_multibase`	`-> str`	`Multibase-encoded (z prefix).`

Contracts:

- `generate()` MUST use the OS CSPRNG (no user-supplied seed).
- `from_private_bytes(kp.private_key_bytes)` MUST produce a functionally equivalent keypair.
- `verify(sign(data), data)` MUST NOT raise.
- `verify(sign(data), different_data)` MUST raise `InvalidSignature`.
- The class MUST be immutable (`frozen=True`).

4.4 X25519 Key Exchange (`EncryptionKeyPair`)

```python
@dataclass(frozen=True)
class EncryptionKeyPair:
    private_key: X25519PrivateKey
    public_key: X25519PublicKey
```

Method/Property	**Signature**	**Description**
`generate`	`classmethod () -> Self`	`Generate from OS CSPRNG.`
`public_key_bytes`	`-> bytes`	`Raw 32-byte public key.`
`public_key_b64url`	`-> str`	`Base64url-encoded public key.`

4.5 ECDH Key Derivation

```python
def ecdh_derive_key(
    our_private: X25519PrivateKey,
    their_public_bytes: bytes,
    info: bytes = HKDF_INFO_AES_KEY,
) -> bytes
```

Behavior:

1. Compute X25519 shared secret.
2. Derive 256-bit key via HKDF-SHA-256 (salt=None, info=`info`).
3. Return 32 bytes.

Contract: If Alice and Bob perform ECDH with each other's keys, they MUST derive the same 32-byte key.

4.6 AES-256-GCM

Function	Signature	Description
`aes_gcm_encrypt`	`(key, plaintext, aad?) -> (nonce, ciphertext)`	Encrypt. Nonce is 12 bytes, randomly generated.
`aes_gcm_decrypt`	`(key, nonce, ciphertext, aad?) -> bytes`	Decrypt. Raises on auth failure.

Contracts:

- Nonce MUST be 12 bytes (96 bits), generated from OS `urandom`.
- `aes_gcm_decrypt(key, *aes_gcm_encrypt(key, pt)) == pt`.
- Decryption with wrong key MUST raise `InvalidTag`.
- Decryption with wrong nonce MUST raise `InvalidTag`.

4.7 Message-Level Encryption

```
python

def encrypt_payload(
    plaintext: bytes,
    recipient_public_key_bytes: bytes,
    info: bytes = HKDF_INFO_AES_KEY,
) -> dict[str, str]
```

Returns: `{"algorithm", "key_exchange", "nonce", "ephemeral_public_key", "ciphertext"}`, all string values.

Behavior:

1. Generate ephemeral X25519 keypair.
2. Derive AES key via ECDH with recipient's public key.
3. Encrypt with AES-256-GCM.
4. Return metadata dict.

```
python

def decrypt_payload(
    encrypted: dict[str, str],
    our_private_key: X25519PrivateKey,
    info: bytes = HKDF_INFO_AES_KEY,
) -> bytes
```

Contract: `decrypt_payload(encrypt_payload(pt, pub), priv) == pt` when `(priv, pub)` form a keypair.

4.8 Identity Derivation

```
python

def derive_agent_id(public_key_bytes: bytes, network: str = "mainnet") ->
str
```

Behavior: `SHA-3-256(public_key_bytes)[:6].hex()` → `did:ocp:<network>:agent-<12 hex chars>`.

Contract: Deterministic — same key always produces the same DID.

4.9 Random Generation

Function	Signature	Description
`generate_nonce`	`(length: int = 32) -> str`	Cryptographic nonce as base64url.
`generate_message_id`	`() -> str`	`msg-<8hex>-<4hex>-<4hex>-<4hex>`.
`generate_uuid_short`	`(prefix: str) -> str`	Prefixed UUID.

5. Agent Identity (`ocp.identity`)

5.1 DID Validation

```
python

def validate_did(did: str) -> bool
```

Pattern: `^did:ocp:[a-z]+:agent-[0-9a-f]{12}$`

5.2 DIDDocument

```
python

@dataclass
class DIDDocument:
    agent_id: str
    public_key_multibase: str
    service_endpoint: str
    capabilities: list[str] = field(default_factory=list)
    trust_attestations: list[dict[str, Any]] = field(default_factory=list)
    revoked_keys: list[dict[str, str]] = field(default_factory=list)
    recovery_keys: list[dict[str, Any]] = field(default_factory=list)
```

Method	Signature	Description
`to_dict`	`() -> dict[str, Any]`	Serialize to W3C-compliant DID Document.
`from_dict`	`classmethod (data: dict) -> DIDDocument`	Deserialize.

`to_dict()` output contract:

- `@context` MUST include `"https://www.w3.org/ns/did/v1"`.
- `verificationMethod[0].type` MUST be `"Ed25519VerificationKey2020"`.
- Recovery keys are appended as additional `verificationMethod` entries with `"purpose": "recovery"`.
- Empty optional lists (`capabilities`, `trust_attestations`, etc.) MUST be omitted from output.

5.3 AgentIdentity

```python
@dataclass
class AgentIdentity:
    signing_keys: SigningKeyPair
    encryption_keys: EncryptionKeyPair
    agent_id: str
    network: str = "mainnet"
    created_at: datetime
    previous_keys: list[SigningKeyPair] = field(default_factory=list)
```

Method	**Signature**	**Description**
generate	classmethod (network?) -> AgentIdentity	Generate new identity.
from_private_key	classmethod (bytes, network?) -> AgentIdentity	Restore from private key (post-recovery).
key_fingerprint	@property -> str	First 16 hex chars of SHA-3-256(public_key).
did_document	(endpoint, capabilities?, recovery_keys?) -> DIDDocument	Build DID Document.
sign	(data: bytes) -> bytes	Sign with current key.
rotate_signing_key	() -> SigningKeyPair	Move current key to previous_keys, generate new one. Returns old key.
rotate_encryption_key	() -> EncryptionKeyPair	Replace encryption key. Returns old key.

`rotate_signing_key()` contract:

- The `agent_id` MUST NOT change (it is derived from the original key, not the current one).
- The old key is appended to `previous_keys`.
- The caller MUST update the DID Document to add the old key to the revocation list after a grace period.

6. Messages (`ocp.messages`)

6.1 MessageType Enum

```python
class MessageType(StrEnum):
    DISCOVERY_PING = "discovery_ping"
    CAPABILITY_QUERY = "capability_query"
    CAPABILITY_RESPONSE = "capability_response"
    KNOWLEDGE_SHARE = "knowledge_share"
    KNOWLEDGE_ACK = "knowledge_ack"
    TASK_REQUEST = "task_request"
    TASK_RESPONSE = "task_response"
    BOND_REQUEST = "bond_request"
    BOND_NEGOTIATE = "bond_negotiate"
    BOND_ACCEPT = "bond_accept"
    BOND_CONFIRM = "bond_confirm"
    BOND_REVOKE = "bond_revoke"
    CONSENSUS_INITIATE = "consensus_initiate"
    CONSENSUS_VOTE = "consensus_vote"
    CONSENSUS_RESULT = "consensus_result"
    BROADCAST = "broadcast"
    RECOVERY_REQUEST = "recovery_request"
    RECOVERY_SHARE_RESPONSE = "recovery_share_response"
    SHARE_REVOKE = "share_revoke"
    ACK = "ack"
    ERROR = "error"
```

21 message types. This enum MUST exactly match the `message_type` enum in `ocpumf.schema.json`.

6.2 MessageBuilder

```Python
class MessageBuilder:
    def __init__(self, sender_id: str, signing_keys: SigningKeyPair) ->
None: ...

    def to(self, receiver_id: str) -> MessageBuilder: ...
    def to_broadcast(self, receiver_id: str = ...) -> MessageBuilder: ...
    def type(self, message_type: MessageType) -> MessageBuilder: ...
    def payload(self, data: dict[str, Any]) -> MessageBuilder: ...
    def priority(self, p: Priority) -> MessageBuilder: ...
    def ttl(self, seconds: int) -> MessageBuilder: ...
```

```
    def tag(self, *tags: str) -> MessageBuilder: ...
    def language(self, lang: str) -> MessageBuilder: ...
    def require_ack(self) -> MessageBuilder: ...
    def correlate(self, message_id: str) -> MessageBuilder: ...
    def trace(self, trace_id: str) -> MessageBuilder: ...
    def encrypt(self, encryption_config: dict[str, Any]) -> MessageBuilder:
...

    def build(self) -> dict[str, Any]: ...
```

`build()` contract:

1. Raise `OCPValidationError` if receiver is not set.
2. Generate a fresh `message_id` via `generate_message_id()`.
3. Set `timestamp` to current UTC time.
4. Construct the full OCPUMF envelope.
5. Compute signature: `Ed25519_Sign(sk, SHA3-256(canonical_json(msg with empty signature)))`.
6. Canonical JSON MUST follow RFC 8785 (JSON Canonicalization Scheme) via the `jsoncanon` library.
7. Set `sender.signature` to the base64url-encoded signature.
8. Return the complete message dict.

`ttl()` contract: Raise `OCPValidationError` if `seconds` is outside `[1, MAX_TTL]`.

6.3 MessageValidator

```
Python

class MessageValidator:
    def validate_structure(self, msg: dict[str, Any]) -> None: ...
    def validate_signature(self, msg: dict[str, Any], public_key_bytes:
bytes) -> None: ...
    def validate(self, msg: dict[str, Any], public_key_bytes: bytes | None =
None) -> None: ...
```

`validate_structure()` checks (in order):

1. All required fields present: `ocp_version`, `message_id`, `timestamp`, `sender`, `receiver`, `message_type`, `payload`.
2. `ocp_version` equals `"1.0"`.

3. `message_id` starts with `"msg-"`.
4. `message_type` is in the valid set.
5. `sender.agent_id` is present.
6. `sender.signature` is present.
7. `receiver.agent_id` is present.
8. `ttl` is in `[1, 86400]`.
9. `timestamp` is a valid ISO 8601 string.

Each check raises `OCPValidationError` with a descriptive message and `code="OCP-400"`.

`validate_signature()` contract:

1. Reconstruct the message with `sender.signature` set to `""`.
2. Canonicalize via RFC 8785.
3. Hash with SHA-3-256.
4. Verify the Ed25519 signature against the provided public key.
5. Raise `OCPAuthError(code="OCP-401")` on failure.

6.4 Utility Functions

Function	Signature	Description
`is_expired`	`(msg: dict) -> bool`	True if `timestamp + ttl < now`.
`build_ack`	`(original_msg, sender_id, signing_keys) -> dict`	ACK for a received message.
`build_error`	`(code, message, ref_id, sender_id, signing_keys, receiver_id, details?) -> dict`	Error response.

7. Knowledge Types (`ocp.knowledge`)

7.1 EmbeddingVector

```
Python

@dataclass
class EmbeddingVector:
    label: str
    vector: bytes           # Raw float array bytes
    source_domain: str = ""
    id: str                 # Auto-generated "emb-<uuid>"
    created_at: datetime
```

Method	Signature	Description
to_dict	() -> dict	Serialize with base64url-encoded vector.
from_floats	staticmethod (label, values: list[float], domain?) -> EmbeddingVector	Create from Python floats (float32 packing).
dimensions	@property -> int	Inferred from len(vector) // 4.

7.2 EmbeddingPackage

```
Python

@dataclass
class EmbeddingPackage:
    dimensions: int
    vectors: list[EmbeddingVector]
    encoding: str = "float32"        # "float32" | "float16" | "bfloat16"
    model_family: str = "transformer"
    normalization: str = "l2"        # "l2" | "none"
```

Validation on `__post_init__`: `encoding` in `{"float32", "float16", "bfloat16"}`, `normalization` in `{"l2", "none"}`, `dimensions >= 1`.

7.3 InsightPackage

```
Python

@dataclass
class InsightPackage:
    topic: str
    description: str
    confidence: float          # [0.0, 1.0]
    source_agent: str          # DID
    derived_from: str
    features: list[InsightFeature] = field(default_factory=list)
    category: str = ""
    evidence_count: int = 0
    recommended_action: str = ""
    false_positive_rate: float | None = None
    methodology: str = ""
```

Validation: `confidence` in `[0.0, 1.0]`, `false_positive_rate` in `[0.0, 1.0]` if set, `topic` non-empty, `source_agent` non-empty.

`to_payload()` contract:

- `knowledge_type` MUST be `"insight"`.
- `anonymized` MUST be `True`. This is hardcoded and not user-configurable.
- `provenance.reproducibility_hash` MUST be the first 32 chars of `SHA-3-256(methodology:topic)` when `methodology` is provided.

7.4 ModelDelta

```
Python

@dataclass
class ModelDelta:
    architecture_family: str
    parameter_count: str
    target_layers: list[str]
    delta_payload: bytes
    source_agent: str
    training_samples: int         # >= 1
    epsilon: float                # (0, MAX_EPSILON]
    dp_delta: float               # (0, 1)
    format: str = "federated_avg" # | "federated_sgd" | "lora_delta" |
"adapter"
    compression: str = "gzip"     # | "none" | "zstd"
    noise_multiplier: float = 1.1
    dp_mechanism: str = "gaussian" # | "laplace"
```

Validation: `epsilon` in `(0, 10.0]`, `dp_delta` in `(0, 1)`, `training_samples >= 1`, `format` / `compression` / `dp_mechanism` in their respective allowed values.

8. Transport (`ocp.transport`)

8.1 TransportConfig

```
Python

@dataclass
class TransportConfig:
    url: str
    transport_type: str = "ocp-ws"
    connect_timeout: float = 10.0
    request_timeout: float = 30.0
    max_retries: int = MAX_RETRY_COUNT
    retry_base_delay: float = RETRY_BASE_DELAY
```

8.2 WebSocketTransport

```
Python

class WebSocketTransport:
    def __init__(self, config, agent_id, signing_keys): ...

    is_connected: bool              # Property
    session_id: str | None          # Property

    async def connect(self) -> None
    async def send(self, message: dict) -> None
    async def receive(self) -> dict
    async def listen(self) -> AsyncIterator[dict]
    def on_message(self, handler: Callable) -> None
    async def close(self) -> None
```

`connect()` contract:

1. Open WebSocket with subprotocol `ocp.v1`.
2. Send `auth_handshake` frame within 5 seconds.
3. Wait for `auth_result` within 5 seconds (`asyncio.wait_for`).
4. Raise `OCPTimeoutError` if handshake times out.
5. Raise `OCPAuthError` if status is `"rejected"`.
6. Set `is_connected = True` and store `session_id`.

`send()` contract: Raise `OCPTransportError` if not connected. Set `is_connected = False` on send failure.

`close()` contract: Idempotent. MUST NOT raise on double-close.

8.3 HTTPTransport

```python
class HTTPTransport:
    def __init__(self, config, agent_id, signing_keys): ...

    async def send(self, message: dict, retry: bool = True) -> dict
    async def close(self) -> None
```

`send()` contract:

1. POST to `config.url` with JSON body, `Authorization: OCP-Ed25519 <agent_id>:<timestamp>:<signature>`, and `X-OCP-Version: 1.0`.
2. On HTTP 202: return `{}`.
3. On HTTP 429: raise `OCPRateLimitError(retry_after=Retry-After header value)`.
4. On HTTP 4xx/5xx: raise `OCPTransportError` with the status code.
5. On network error with `retry=True`: retry with exponential backoff (1s, 2s, 4s, 8s, 16s), up to `max_retries`. Raise after all retries exhausted.
6. HTTP/2 SHOULD be used when available.

9. Registry Client (`ocp.registry`)

9.1 RegistryClient

```
Python

class RegistryClient:
    def __init__(self, config: RegistryConfig | None = None): ...

    async def register(self, record: AgentRecord, signature: str) -> dict
    async def discover(self, domains?, capabilities?, min_trust_level?,
status?, limit?, offset?) -> list[dict]
    async def resolve(self, agent_id: str) -> dict
    async def deregister(self, agent_id: str, signature: str) -> dict
    async def close(self) -> None
```

`resolve()` contract: Raise `OCPAgentNotFoundError` on HTTP 404. Raise `OCPTransportError` on other failures.

`discover()` contract: Return empty list on no matches (MUST NOT raise). Raise `OCPTransportError` on network failure.

10. Privacy Validation Layer (`ocp.pvl`)

10.1 PVLResult

```
Python

@dataclass(frozen=True)
class PVLResult:
    passed: bool
    rejection_code: str | None = None
    rejection_reason: str | None = None
```

10.2 Validation Function

```
Python

def validate_knowledge_payload(
    payload: dict[str, Any],
    max_payload_bytes: int = MAX_PAYLOAD_SIZE,
) -> PVLResult
```

Check execution order (first failure wins):

1. **PVL-005:** Payload JSON byte size > `max_payload_bytes`.
2. **PVL-004:** Missing provenance (for `insight` and `model_delta` types).
3. **PVL-002:** `anonymized != True` (for `insight` type).
4. **PVL-003:** Missing/invalid differential privacy parameters (for `model_delta` type), including `epsilon > MAX_EPSILON`.
5. **PVL-001:** PII detected in any string value (recursive deep scan).

PII detection contract:

The reference implementation scans using regex patterns for: SSN, email addresses, phone numbers, credit card numbers, IP addresses, UK postcodes, and Japanese phone numbers. The scanner recursively extracts all string values from nested dicts, lists, and tuples up to a maximum depth of 20.

Production implementations SHOULD augment regex scanning with a trained NER model.

10.3 Enforcement Function

```
Python

def enforce_pvl(payload: dict, max_payload_bytes: int = MAX_PAYLOAD_SIZE) ->
None
```

Calls `validate_knowledge_payload` and raises `OCPPrivacyViolation(pvl_code=...)` on failure. Returns `None` on success.

11. Consensus (`ocp.consensus`)

11.1 ConsensusConfig

```python
@dataclass
class ConsensusConfig:
    topic: str
    options: list[str]            # >= 2 unique values
    min_participants: int = 5
    threshold: float = 0.67       # (0.0, 1.0]
    weighted: bool = True
    deadline: datetime | None = None
    min_trust_level: int = 2
    required_domains: list[str] = field(default_factory=list)
```

Validation: `len(options) >= 2`, `threshold` in `(0.0, 1.0]`, `min_participants >= 1`.

11.2 Vote

```python
@dataclass
class Vote:
    voter_id: str
    option: str
    confidence: float             # [0.0, 1.0]
    trust_score: float            # [0.0, 1.0]
    signature: str = ""
    timestamp: datetime = field(default_factory=...)
```

Method	Signature	Description
sign	(signing_keys, consensus_id) -> None	Sign the vote content.

11.3 ConsensusRound

```
Python

class ConsensusRound:
    consensus_id: str              # Auto-generated "con-<uuid>"
    config: ConsensusConfig

    vote_count: int                # Property
    is_past_deadline: bool         # Property

    def initiation_payload(self) -> dict: ...
    def cast_vote(self, vote: Vote) -> bool: ...
    def resolve(self) -> ConsensusResult: ...
    def result_payload(self, result: ConsensusResult) -> dict: ...
```

`cast_vote()` contract — returns `False` (rejection) when:

- Duplicate `voter_id` (already voted).
- `option` not in `config.options`.
- `confidence` outside `[0.0, 1.0]`.
- Deadline has passed.

`resolve()` contract:

```
Python

For each option:
    if weighted:
        score = Σ(voter.trust_score × voter.confidence) for voters who
selected this option
    else:
        score = count of voters who selected this option

total = sum of all scores
reached_quorum = vote_count >= min_participants
reached_threshold = (max_score / total) >= threshold

winner = best_option if (reached_quorum AND reached_threshold) else None
```

11.4 ConsensusResult

```
Python

@dataclass(frozen=True)
```

```
class ConsensusResult:
    consensus_id: str
    winner: str | None
    weighted_scores: dict[str, float]
    total_votes: int
    reached_quorum: bool
    reached_threshold: bool
```

12. Trust Model (`ocp.trust`)

12.1 TrustLevel Enum

```
Python

class TrustLevel(IntEnum):
    ANONYMOUS = 0
    IDENTIFIED = 1
    VOUCHED = 2
    BONDED = 3
    CERTIFIED = 4
```

12.2 Vouch

```
Python

@dataclass
class Vouch:
    attester_id: str
    subject_id: str
    domains: list[str]
    signing_keys: SigningKeyPair | None = None
    issued_at: datetime
    expires_at: datetime | None   # Default: issued_at + 365 days
    revoked: bool = False
```

Validation:

- `attester_id == subject_id` → raise `ValueError` ("Self-vouching is prohibited").
- `expires_at - issued_at > 365 days` → raise `ValueError`.

- `expires_at <= issued_at` → raise `ValueError`.

Property	Type	Description
`is_expired`	`bool`	`now() > expires_at`
`is_valid`	`bool`	`not is_expired and not revoked`

12.3 BondPermissions

```python
@dataclass
class BondPermissions:
    knowledge_share: bool = True
    knowledge_allowed_types: list[str] = ["insight", "embedding"]
    max_payload_bytes: int = 10_485_760
    task_delegate: bool = True
    max_concurrent_tasks: int = 5
    task_timeout_seconds: int = 300
    model_delta_share: bool = False
```

Method	Signature	Description
`to_dict`	`() -> dict`	Serialize.
`from_dict`	`classmethod (dict) -> BondPermissions`	Deserialize.
`intersect`	`(other: BondPermissions) -> BondPermissions`	Compute intersection (AND booleans, MIN numerics, intersect lists).

12.4 Bond

```python
@dataclass
class Bond:
    agent_a: str
    agent_b: str
    permissions: BondPermissions
```

```
    bond_id: str                    # Auto-generated
    established_at: datetime
    expires_at: datetime | None     # Default: established_at + 180 days
    renewal: str = "manual"         # "auto" | "manual" | "none"
    revoked: bool = False
    revoked_by: str | None = None
```

Validation: expires_at - established_at > 365 days → raise ValueError.

Method/Property	**Type**	**Description**
is_expired	bool	now() > expires_at
is_active	bool	not is_expired and not revoked
permits_knowledge_share(type)	bool	Active + knowledge_share enabled + type in allowed_types
permits_task_delegation()	bool	Active + task_delegate enabled
permits_model_delta()	bool	Active + model_delta_share enabled
revoke(revoked_by)	None	Set revoked=True, revoked_by=agent_id
involves(agent_id)	bool	Agent is party to bond
peer_of(agent_id)	str	Other party's DID. Raises ValueError if not involved.

12.5 Trust Score

```
Python

def compute_trust_score(
    is_verified: bool,
    vouch_count: int,
    bond_count: int,
    interaction_reputation: float,
    uptime_ratio: float,
    weights: TrustScoreWeights | None = None,
) -> float
```

Formula:

```
score = w1 * (1.0 if verified else 0.0)
```

```
      + w2 * min(vouch_count / 10, 1.0)
      + w3 * min(bond_count / 5, 1.0)
      + w4 * clamp(interaction_reputation, 0, 1)
      + w5 * clamp(uptime_ratio, 0, 1)
```

Default weights: `(0.20, 0.25, 0.25, 0.20, 0.10)`. Weights MUST sum to 1.0 (± 0.001). Output is clamped to `[0.0, 1.0]` and rounded to 4 decimal places.

```
Python

def determine_trust_level(
    is_verified: bool,
    vouch_count: int,
    bond_count: int,
    is_certified: bool = False,
) -> TrustLevel
```

Logic:

```
if is_certified:      return CERTIFIED
if bond_count >= 1 and vouch_count >= 3: return BONDED
if vouch_count >= 3:  return VOUCHED
if is_verified:       return IDENTIFIED
return ANONYMOUS
```

13. Key Recovery (`ocp.recovery`)

13.1 Shamir's Secret Sharing

```
Python

def split_secret(
    secret: bytes,
    threshold: int = 3,
    num_shares: int = 5,
) -> list[tuple[int, bytes]]
```

Parameters: `threshold >= 2`, `num_shares >= threshold`, `num_shares <= 255`, `secret` non-empty.

Algorithm: For each byte, construct a random polynomial of degree `threshold - 1` over GF(2^8) (AES polynomial `0x11B`), evaluate at points `x ∈ {1, ..., num_shares}`.

Contract: For any subset of `t` shares: `reconstruct_secret(subset) == secret`. For any subset of `t-1` shares: reconstruction MUST produce incorrect output with overwhelming probability.

```
Python

def reconstruct_secret(shares: list[tuple[int, bytes]]) -> bytes
```

Algorithm: Lagrange interpolation at `x=0` over GF(2^8), per-byte.

Validation: `len(shares) >= 2`, no duplicate indices, all shares same length.

13.2 RecoveryManager

```
Python

class RecoveryManager:
    def __init__(self, agent_id: str, signing_keys: SigningKeyPair): ...

    key_fingerprint: str          # Property: SHA-3-256(pub_key)[:16] hex

    def generate_shares(self, threshold=3, num_shares=5) ->
list[RecoveryShare]
    def encrypt_share_for_custodian(self, share, custodian_pub_bytes) ->
dict[str, str]
    @staticmethod
    def decrypt_share(encrypted: dict, custodian_private_key) -> bytes
    @staticmethod
    def reconstruct(shares: list[tuple[int, bytes]], expected_fingerprint:
str) -> bytes
    def build_recovery_request_payload(self, ephemeral_key, secondary_key?)
-> dict
```

`reconstruct()` contract:

1. Call `reconstruct_secret(shares)`.
2. Derive Ed25519 public key from reconstructed private key.
3. Compute `SHA-3-256(public_key)[:16].hex()`.

4. Compare to `expected_fingerprint`.
5. Raise `OCPRecoveryError` on mismatch.
6. Return 32-byte private key on success.

14. CLI Tools (`ocp.cli`)

14.1 `ocp-keygen`

```
usage: ocp-keygen [--network {mainnet,testnet}] [--endpoint URL] [--json]
```

Behavior:

1. Generate a new `AgentIdentity`.
2. Print agent ID, network, public key, key fingerprint, encryption public key.
3. If `--endpoint` provided, also print the full DID Document.
4. If `--json`, output as JSON instead of human-readable text.
5. MUST NOT print the private key.

15. Compliance Checker (`ocp.compliance`)

15.1 `ocp-compliance`

```
usage: python -m ocp.compliance
```

Test suite (6 categories):

Category	Tests
Identity & DID	DID format, uniqueness, DID Document structure, W3C context
Cryptographic Primitives	Ed25519 sign/verify, ECDH agreement, AES-GCM roundtrip, base64url roundtrip, SHA-3 determinism
Message Building & Validation	Message construction, signing, structural validation, signature verification
Privacy Validation Layer	PII detection (email, SSN, phone), anonymization enforcement, DP parameter validation, provenance check
Key Recovery (Shamir SSS)	3-of-5 reconstruction, all subsets, wrong share rejection
Consensus Protocol	Basic consensus, duplicate vote rejection, quorum enforcement

Exit codes: `0` = all passed, `1` = one or more failed.

16. Concurrency Model

16.1 Async-First Design

All I/O-bound operations (transport, registry, database) are `async`. The SDK is designed for use with `asyncio`. There is no synchronous API.

16.2 Thread Safety

Component	Thread Safe	Notes
`SigningKeyPair`	Yes	Immutable (frozen dataclass)
`EncryptionKeyPair`	Yes	Immutable
`AgentIdentity`	No	Mutable (`previous_keys` list)
`Agent`	No	Mutable internal state (bonds, vouches, transports)
`MessageBuilder`	No	Stateful builder pattern
`MessageValidator`	Yes	Stateless
`ConsensusRound`	No	Mutable vote collection
`RecoveryManager`	Yes	Stateless after construction
PVL functions	Yes	Pure functions
Crypto functions	Yes	Pure functions

Applications requiring concurrent access to an `Agent` MUST use external synchronization (e.g., `asyncio.Lock`).

16.3 Resource Cleanup

All classes with async resources (`Agent`, `WebSocketTransport`, `HTTPTransport`, `RegistryClient`) implement `async close()`. Callers SHOULD use `try/finally` or `contextlib.asynccontextmanager` to ensure cleanup.

17. Logging

The SDK uses Python's standard `logging` module with the logger name `"ocp"` and per-module sub-loggers (`"ocp.transport"`, `"ocp.agent"`, etc.).

Rules:

- `DEBUG`: Message-level tracing, cryptographic operation details.
- `INFO`: Lifecycle events (connected, registered, bond established, knowledge shared).
- `WARNING`: Recoverable failures (duplicate message, expired vouch, PVL rejection).
- `ERROR`: Unrecoverable failures (transport error, handler crash).
- The SDK MUST NOT configure logging handlers, formatters, or levels. Configuration is the application's responsibility.
- The SDK MUST NOT use `print()`.

18. Dependencies

18.1 Required

Package	Version	Purpose
`cryptography`	≥ 43.0	Ed25519, X25519, AES-GCM, HKDF
`pynacl`	≥ 1.5.0	Supplementary NaCl bindings
`websockets`	≥ 13.0	WebSocket transport
`httpx`	≥ 0.27	HTTP transport (HTTP/2 support)
`pydantic`	≥ 2.7	Data validation (internal)
`jsonschema`	≥ 4.22	Schema validation
`jsoncanon`	≥ 0.2	RFC 8785 canonical JSON for signing

18.2 Development

Package	Version	Purpose
`pytest`	≥ 8.0	Testing
`pytest-asyncio`	≥ 0.24	Async test support
`pytest-cov`	≥ 5.0	Coverage reporting
`ruff`	≥ 0.4	Linting and formatting
`mypy`	≥ 1.10	Static type checking

18.3 Dependency Policy

- The SDK MUST NOT depend on any ML/AI framework (PyTorch, TensorFlow, etc.).
- The SDK MUST NOT depend on any database library.
- The SDK MUST NOT depend on any web framework.
- All cryptographic operations MUST be delegated to `cryptography` or `pynacl`. The SDK MUST NOT implement any cryptographic algorithm.
- The GF(2^8) arithmetic in `recovery.py` is the sole exception — it implements finite field operations for Shamir's Secret Sharing because no standard library provides this.

19. Conformance

19.1 Conformance Levels

Level	Requirements
OCP Core SDK	Modules: `crypto`, `identity`, `messages`, `exceptions`, `constants`. Passes: Identity, Crypto, and Message compliance tests.
OCP Knowledge SDK	OCP Core SDK + modules: `knowledge`, `pvl`. Passes: Knowledge and PVL compliance tests.

OCP Full SDK	All modules. Passes: All compliance tests. Coverage ≥ 85%.

The reference Python SDK targets **OCP Full SDK** conformance.

19.2 Compliance Test Matrix

Test Category	# Tests	Required For
Identity & DID	6	Core
Cryptographic Primitives	12	Core
Message Building & Validation	8	Core
Knowledge Types	6	Knowledge
Privacy Validation Layer	10	Knowledge
Trust & Bonding	8	Full
Consensus	6	Full
Key Recovery	10	Full
Transport	4	Full
Total	**70**	—

19.3 Backward Compatibility

Within a major version:

- No public symbol may be removed.
- No method signature may have required parameters added.
- No method may change its return type.
- No exception type may be changed for a given failure mode.
- New optional parameters may be added to methods.
- New public symbols may be added.
- New exception subtypes may be added.

Deprecated symbols MUST emit `DeprecationWarning` for at least 2 minor releases before removal in the next major version.

OpenCognition Protocol (OCP) - Disclosure and Disclaimer

Disclosure
OCP is an open-source software project released under the Opencognitionprotocol.org, MIT, GPL v3 license. It is developed and maintained by a community of contributors. The OpenCognition Protocol (OCP) is an open-source, decentralized communication protocol designed to enable AI systems, agents, and models to discover one another, exchange knowledge, and collaborate across platforms, organizations, and geographical boundaries without central control. Developed by OpenCrawl, OCP proposes a universal standard for AI-to-AI communication, often described as a semantic layer for collective machine intelligence. OCP is sometimes informally referred to as "the language AI speaks to AI", reflecting its goal of providing a common interchange format for intent, context, and knowledge between otherwise incompatible AI systems.

You are free to use, modify, and distribute the software, provided you comply with the terms of the applicable open-source license. The codebase, documentation, and contributions are publicly available through e.g., GitHub or GitLab

Disclaimer
OCP is provided "as-is" and without any warranties, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose, and non-infringement. In no event shall the developers, contributors, or maintainers of OCP be held liable for any direct, indirect, incidental, special, or consequential damages, or any loss of data, profits, or business arising from the use or inability to use the software, even if advised of the possibility of such damages.

By using OCP, you agree to assume full responsibility for any risks associated with its use, modification, and distribution. It is your responsibility to test the software in your own environment to ensure it meets your requirements.

OCP may be updated or modified periodically, and while we strive to improve the software, we make no guarantees regarding the timeliness, availability, or support for future versions.

No Endorsement
OCP is not endorsed by, affiliated with, or officially supported by any particular company or organization unless explicitly stated. Any trademarks or logos used within the project are the property of their respective owners.

Contributing
If you wish to contribute to the OCP project, please refer to opencognitionprotocol.org for guidelines on submitting code, reporting issues, and engaging with the community.

License
This project is licensed under the opencognitionprotocol.org License.

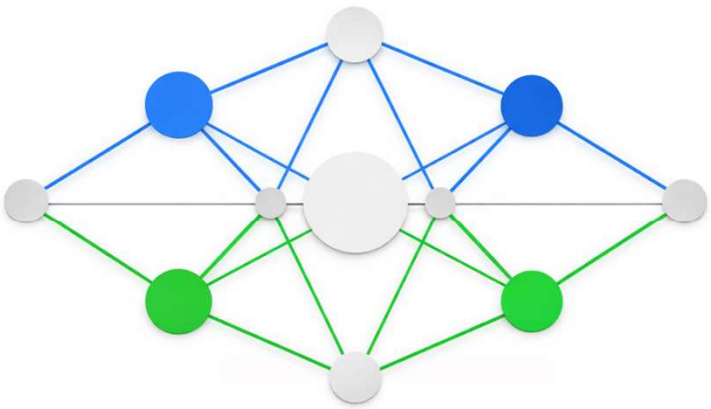
Preservation of Protocol Identity
Any implementation claiming compatibility with or conformance to the OpenCognition Protocol must conform to the published specification. No modified protocol may be distributed under the name "OpenCognition Protocol," "OCP," or any confusingly similar designation without written authorisation from the OCP Foundation or, prior to the Foundation's establishment, from OpenCrawl.

Prohibited Uses
A Deploying Organisation must not use OCP to:

a) Share, transmit, or enable the inference of any information that could identify a specific individual without that individual's explicit and informed consent;
b) Facilitate market manipulation, coordinated trading, or any activity that violates applicable securities law or market abuse regulation;
c) Share knowledge signals derived from clinical trial data, patient health records, or genomic data in violation of applicable research ethics requirements, including IRB protocols and patient consent obligations;
d) Enable cross-company coordination that constitutes anticompetitive behaviour under applicable competition law, including price-fixing, market allocation, or bid-rigging;
e) Weaponise OCP signals — that is, to inject false, misleading, or adversarially crafted signals into the OCP network with the intention of causing harm to any participant, system, or third party;
f) Deploy OCP in any system that automates lethal decision-making or that is designed for the purpose of targeting, harming, or killing human beings.

No Warranty
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Limitation of Liability
TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL OPENCRAWL, THE OCP FOUNDATION, OR ANY CONTRIBUTOR BE LIABLE FOR ANY INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, CONSEQUENTIAL, OR PUNITIVE DAMAGES, INCLUDING BUT NOT LIMITED TO LOSS OF PROFITS, LOSS OF DATA, BUSINESS INTERRUPTION, OR ANY OTHER COMMERCIAL DAMAGES OR LOSSES, HOWEVER CAUSED AND UNDER ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT, ARISING OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THIS LICENSE AGREEMENT.



Where Minds Meet Machines

OpenCognitionProtocol.org