# Intent-Driven 6G Service Orchestration: Grounded Translation, Validation, and Decomposition

**Anonymous Authors**[1]

## Abstract

Intent-based automation for 6G envisions networks steered by high-level goals rather than low-level configurations. Existing LLM-based approaches translate natural language into plausible intent representations but typically omit what production deployment requires: grounding in actual service catalogs, formal validation, and cross-layer decomposition. We address this with an agentic workflow comprising three coupled reasoning layers: (i) *grounding* the translation in a semantic service catalog that exposes TMF-compliant service specifications; (ii) *validation* of the RDF intent via SHACL structural checking against the TMF Intent Ontology; and (iii) *decomposition* that selects a CFSS profile via constraint satisfaction over QoS capability envelopes, then covers its infrastructure requirements with RFSS profiles via weighted set cover. Across 930 benchmark runs over six GPT-4.1/5 models, the workflow achieves 97% success in structured mode and 90% on average across natural-language scenarios, with 100% correct rejection of infeasible requests. Grounding LLM context in catalog capability metadata reduces adversarial hallucinations by 26 percentage points; larger gains than scaling model size alone.

## 1. Introduction

The evolution toward 6G demands intent-driven service orchestration: applications and operators express high-level service requirements and the management system validates, decomposes, and fulfills them as formal service orders (Clemm et al., 2022; Leivadeas & Falkner, 2023). The TM Forum defines two complementary frameworks for this pipeline: the Shared Information and Data (SID) model,[1] which structures service catalogs into customer-facing service specifications (CFSSs) and resource-facing service specifications (RFSSs) with explicit QoS capability envelopes, and the Intent Common Model (TM Forum, 2024a), which standardizes intent expression as RDF graphs over the TMF Intent Ontology (TIO), aligned with ETSI ZSM (ETSI, 2019) and 3GPP management frameworks. Bridging these two, turning an intent into a validated service order against the SID catalog; aligns with the foreseen evolution toward intent-driven autonomous networks (Niemöller et al., 2024), yet remains largely unsolved.

As Ficzere et al. (2025) argue, current work on Generative AI for IBN remains narrowly focused on intent *translation*, leaving substantial gaps across the rest of the IBN life cycle, notably feasibility validation, orchestration composition, and structured reporting. Recent large language model (LLM)-based approaches to intent and slice management (Mekrache et al., 2025; 2024; Guo et al., 2025) generate intent expressions from natural language, but typically skip three capabilities that are essential for telecom service management (Niemöller et al., 2024): *structural validation* of the generated intent against the TIO, *feasibility reasoning* against the QoS capability envelopes declared in the SID catalog, and *cross-layer decomposition* of customer-facing selections into resource-facing infrastructure (network slice, edge compute, transport). Without all three, an LLM can produce syntactically plausible but semantically invalid or infeasible service orders that no intent handler should accept.

To illustrate, consider an operator requesting "cloud gaming with latency $\leq 10$ ms." Under this model, the intent triggers a service order (TM Forum, 2024c) against the service catalog (TM Forum, 2024b): the system should (1) generate a structurally valid RDF intent conforming to TIO; (2) discover that the Pro Gaming CFSS (latency envelope 5–10 ms) satisfies the constraint while Casual Gaming (50–100 ms) does not; and (3) decompose Pro Gaming's infrastructure requirements into RFSSs: a URLLC network-slice profile, an edge-compute pool, and a transport segment. This pipeline

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

[1]https://www.tmforum.org/open-digital-architecture/information-framework-sid/

goes well beyond intent translation: it requires grounding, validation, and decomposition as first-class concerns.

We present an agentic system that addresses this end-to-end. It sits at the natural-language boundary of a TMF intent-management hierarchy (TM Forum, 2022), produces validated RDF intents, and emits a CFSS selection together with an RFSS decomposition that downstream intent handlers can consume. The system does *not* attempt to configure RAN or transport equipment directly; instead, it grounds LLM generation in a semantic service catalog and a formal RequirementCapability model, so that "what" is requested and "what can be delivered" are explicitly reconciled before a plan leaves the NL layer.

## 2. Background

### 2.1. TMF Intent Ontology

The TIO (TM Forum, 2024a) is the RDF vocabulary defined for expressing network management intents. It is specified across 15 ontology modules covering intent structure, logical composition, quantity constraints, metrics, and lifecycle management. The core constructs include `icm:Intent` as the top-level container, expectations for requirements (specialized into delivery, property, and reporting), and targets for scope. Logical operators (`allOf`, `anyOf`) compose expectations into boolean expressions, and the Quantity Ontology provides metric-based constraints (`atLeast`, `atMost`) with unit handling for properties such as latency. Listing 1 shows a minimal intent requesting cloud gaming with latency $\leq 10$ ms.

*Listing 1.* TIO intent: cloud gaming with latency $\leq 10$ ms.

```
@prefix icm: <http://tio.../IntentCommonModel/> .
@prefix log: <http://tio.../LogicalOperators/> .
@prefix met: <http://tio.../MetricsAndObservations/> .
@prefix quan: <http://tio.../QuantityOntology/> .
@prefix dim: <http://tio.../Dimensions/> .

ex:GamingIntent a icm:Intent ;
   log:allOf (ex:DeliveryExp ex:LatencyExp ) .

ex:DeliveryExp a icm:DeliveryExpectation ;
   icm:target [a icm:Target ];
   icm:deliveryType cfss:GamingCFSS .

ex:LatencyCond a log:Condition ;
   quan:atMost ([met:lastValue (dim:Latency ) ]
             "10ms"^^quan:quantity ) .

ex:LatencyExp a icm:PropertyExpectation ;
   log:allOf (ex:LatencyCond ) .
```

Composing such intents correctly is error-prone due to TIO's recursive logical operators, quantity expressions, and cross-expectation dependencies; guaranteeing ontology compliance before an intent reaches a handler is therefore essential. We leverage TIO-SHACL (Martins et al., 2026) to provide structural validation against the ontology.

### 2.2. SID Service Catalog

The TM Forum Information Framework (SID) is a comprehensive reference model that organizes telecom concepts across product, service, and resource domains. In this work we focus on the *service domain* used by service catalogs: CFSSs define application-facing service profiles with QoS capability envelopes visible to customers and operators (e.g., as exposed through TMF 633/641), while RFSSs describe the underlying infrastructure capabilities (network slices, compute pools, transport segments) required to deliver those services. A CFSS profile typically requires several RFSS profiles working together; resolving this many-to-many mapping between layers is precisely the decomposition problem we address. In ODA terms, our system sits between the intent interface and TMF 633/641-driven service order management.

## 3. Semantic Service Catalog and Problem Formulation

To enable formal reasoning over the SID service domain, we build a semantic layer that makes QoS capabilities and requirements explicit and machine-actionable. This layer serves as the grounding infrastructure for the agentic pipeline: it provides the formal substrate against which an LLM-generated intent can be validated for feasibility, matched to a CFSS profile, and decomposed into RFSS selections. Figure 1 illustrates the resulting catalog hierarchy, where CFSS profiles declare QoS capability envelopes and RFSS profiles provide the underlying infrastructure.

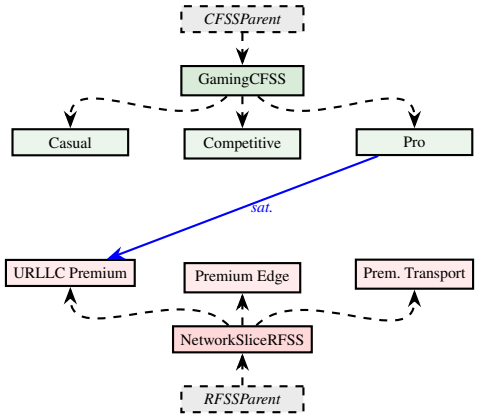


*Figure 1.* Semantic service catalog hierarchy. CFSS profiles (green) declare QoS capability envelopes; RFSS profiles (red) provide infrastructure. The blue arrow denotes a satisfaction relationship computed by the matching algorithm.

### 3.1. Requirement-Capability ontology

The RequirementCapability model introduces two dual predicates over this hierarchy (Figure 1). The *provides* predicate

declares what a profile can deliver for a given metric (a range with minimum, maximum, unit, or an enumeration such as URLLC/eMBB). The *requires* predicate declares what an intent or CFSS profile demands from the layer below. Each layer acts as both consumer and provider: an intent requires CFSS capabilities; a CFSS profile provides capabilities to intents and requires RFSS capabilities; RFSS profiles provide capabilities to CFSS profiles.

A *capability* $c \in C(p)$ for profile $p$ specifies a metric with a deliverable range $[c^-, c^+]$ and unit, or an enumeration of allowed values. A *requirement* is defined analogously. Appendix B shows the RDF serialization of these predicates.

### 3.2. Problem Formulation

Given a natural-language intent and a service catalog, the intent-driven service management problem decomposes into four sub-problems. Structural correctness of the generated RDF intent (Shapes Constraint Language (SHACL) validation against TIO) is a prerequisite enforced at generation time; the sub-problems below concern the end-to-end pipeline from natural language to validated service order.

**P0 (Intent Translation).** Generate a structurally valid TIO/RDF intent from a natural-language requirement, passing SHACL validation.

**P1 (Constraint Extraction).** Extract from the validated intent a structured set of QoS requirements $R = \{r_1, \ldots, r_n\}$ where each $r_i = (m_i, op_i, v_i, u_i)$ is a tuple of metric, operator, value and unit (or $(m_i, op_i, e_i)$ for enumerations).

**P2 (CFSS Selection).** Given requirements $R$ and CFSS profiles $P = \{p_1, \ldots, p_m\}$ with capabilities $C(p_j)$, find a minimum-cost profile satisfying all requirements:

$$p^* = \arg\min_{p \in P} \; cost(p) \;\; \text{s.t.} \;\; \forall\, r \in R,\; \exists\, c \in C(p) \colon \mathrm{Sat}(c, r). \tag{1}$$

**P3 (RFSS Decomposition).** Given the selected CFSS profile's infrastructure requirements $R_{\text{cfss}}$ and candidate RFSS profiles $Q$ (network-slice, edge, transport) with capabilities $C(q_i)$ and costs $w(q_i)$, find the minimum-cost subset covering all requirements:

$$S^* = \arg\min_{S \subseteq Q} \; \textstyle\sum_{q \in S} w(q) \;\; \text{s.t.} \;\; \bigcup_{q \in S} C(q) \supseteq R_{\text{cfss}}. \tag{2}$$

P2 is a capability-set constraint-satisfaction problem over profile choice; P3 is weighted set cover over infrastructure profiles, which we solve with a greedy heuristic ($O(\ln n)$ approximation) and compare against a CP-SAT solver.

### 3.3. Two-Layer Validation

The system combines two validation layers. *SHACL shapes* (Martins et al., 2026) validate structural correctness of the intent RDF against TIO: required fields, correct types, and cardinality. The *RequirementCapability* layer provides feasibility reasoning: range- and enumeration-based predicates over capability sets determine whether any catalog profile can satisfy the extracted requirements. SHACL catches missing or malformed constructs (e.g., a missing metric field), while RequirementCapability catches semantic infeasibility (e.g., requesting a QoS level no profile can deliver). This separation enables the system to distinguish fixable structural errors from fundamentally infeasible requirements.

## 4. Agentic Architecture

The system is implemented as a LangGraph[2] state machine that orchestrates LLM calls and external tool invocations. Rather than a RAN or transport control-plane component, the system sits at the NL boundary of a TMF-style intent-management hierarchy: it ingests a natural-language requirement, grounds it in the semantic service catalog, emits a validated TIO/RDF intent together with a CFSS selection and an RFSS decomposition plan, and hands the result to downstream intent handlers for execution. 3GPP RAN and transport controllers remain out of scope.

### 4.1. Dual-Mode Workflow

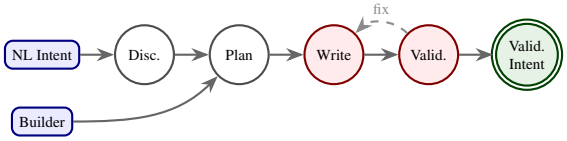


*Figure 2.* Dual-mode agentic workflow. NL mode includes catalog discovery; Builder mode bypasses directly to planning. Both converge at generation and validation with a self-healing loop (dashed).

The system supports two entry modes (Figure 2). **NL mode** processes free-text requirements through discovery (semantic search over the catalog to identify candidate CFSS families), capability retrieval, and plan decomposition. **Builder mode** accepts structured inputs specifying the target CFSS profile and requirement set directly, bypassing NL processing for programmatic integration with reduced token consumption. Both modes converge on the same plan structure that drives intent generation and validation.

### 4.2. Intent Generation and Self-Healing

The write-intent stage transforms the plan into RDF/Turtle. The prompt combines a stable prefix containing TIO syntax rules, conditionally injected validated patterns retrieved by fuzzy matching, and a dynamic context comprising the

[2] https://github.com/langchain-ai/langgraph

**Algorithm 1** Grounded intent composition. Components defined in Section 3.

**Require:** Requirement $\rho$ (NL or structured), catalog $(P, Q)$, shapes $\Sigma$
**Ensure:** Validated intent $I$ with CFSS $p^*$ and RFSS cover $S^*$, or $\perp$

1: $\mathcal{M} \leftarrow \textsc{Ground}(\rho, P)$ {available-property metadata}
2: $I \leftarrow \textsc{Retry}_k\big[\textsc{LLMWrite}(\rho, \mathcal{M}, \Sigma)$ ST $\textsc{Shacl}(I, \Sigma)\big]$
3: **if** $I = \perp$ **then**
4: **return** $\perp$ {structural repair failed}
5: **end if**
6: $R \leftarrow \textsc{Extract}(I)$ {P1: constraint set}
7: $p^* \leftarrow \textsc{SatCfss}(R, P)$ {P2: CSP, Eq. 1}
8: **if** $p^* = \perp$ **then**
9: **return** $\perp$ {infeasible: no CFSS covers $R$}
10: **end if**
11: $S^* \leftarrow \textsc{CoverRfss}(R_{\text{cfss}}(p^*), Q)$ {P3: Eq. 2}
12: **return** $(I, p^*, S^*)$

requirement, available-property metadata from the catalog, and the plan. Available-property metadata extracted from the catalog's `req:provides` blocks is injected explicitly so that the model is told which QoS metrics the target service models (see Section 5).

After generation, the RDF passes through a two-stage validation pipeline with automatic error correction. Algorithm 1 shows the end-to-end composition: LLM-based NL-to-RDF translation (P1), SHACL structural validation, CSP-style CFSS selection (P2), and weighted-set-cover RFSS decomposition (P3) are composed as a single pipeline, with a bounded retry combinator $\textsc{Retry}_k$ providing self-healing around the fallible LLM step. The routing logic distinguishes three outcomes: both checks pass (accept the intent); SHACL fails (retry generation with violation context); feasibility fails (terminate—the intent is structurally valid but cannot be satisfied by any catalog profile).

$\textsc{Retry}_k$ invokes LLMWrite up to $k$ times (we use $k = 5$ for both syntactic and semantic correction), each retry seeded with the prior SHACL violation report and with validated patterns retrieved via fuzzy matching. This mechanism is critical for deployability: rather than silently producing an invalid or infeasible intent, the system either converges to a valid one or explicitly terminates with a diagnosis (line 3 for unrepairable SHACL violations, line 6 for CFSS-level infeasibility).

## 5. Experimental Evaluation

We evaluate the system across six LLM models using five benchmark configurations totaling 930 runs. The evaluation measures end-to-end success rates, per-phase pipeline accuracy, and the impact of prompt difficulty and context grounding on model performance.

### 5.1. Setup

The six models span two Azure OpenAI families, GPT-4.1 (nano, mini, full) and GPT-5 (nano, mini, full), each executed five times per input at temperature 0.0. The catalog contains five CFSS families (Cloud Gaming, Video Streaming, Video Conferencing, Audio, Data Connectivity) with multiple profiles per family, plus the three RFSS categories described in Section 3 (network-slice, edge-compute, transport). Profiles declare QoS capability envelopes over metrics such as latency, jitter, throughput, bandwidth, packet loss, availability, and enumeration metrics such as slice type (URLLC/eMBB/mMTC) and 3GPP 5QI.

The benchmark configurations form two axes. The first axis is *input mode*: NL (free-text requirement, full pipeline) vs. Builder (structured input, direct plan). The second axis is *difficulty*: constrained, adversarial, infeasible.

| Config | Mode | Runs | Description |
|---|---|---|---|
| CFSS | Builder | 150 | Structured input, write+validate only |
| RFSS | Builder | 270 | Decomposition targets (slice, edge, transport) |
| Adversarial | Builder | 120 | Additional validity/reporting clauses |
| Infeasible | Builder | 90 | Impossible QoS values |
| Constrained | NL | 150 | Valid constraints on modeled metrics |
| Adversarial | NL | 120 | References to unmodeled metrics |
| Infeasible | NL | 90 | Values exceeding all profile envelopes |

*Table 1.* Benchmark configurations (930 total runs).

End-to-end NL evaluation targets CFSS selection, constraint extraction, and RDF generation; RFSS decomposition is evaluated through Builder RFSS (structured) and, separately, through the CP-SAT vs. greedy algorithmic benchmark described in Appendix D.

**Success criteria.** A run is successful when the pipeline produces the expected terminal outcome: for Builder and NL Constrained/Adversarial, a structurally valid RDF intent passing both SHACL and feasibility checks *and* selecting the ground-truth profile (exact child class when the prompt names a specific profile, any class within the same CFSS family when the prompt uses only numeric constraints); for NL Infeasible, explicit rejection with an infeasibility diagnosis ($\perp$ in Algorithm 1). Silently producing a valid-looking but incorrect intent counts as a failure in all scenarios.

### 5.2. End-to-End Results

Table 2 presents per-model success rates. Builder mode is strong overall (97% CFSS / 100% RFSS), with GPT-4.1-nano's 84% CFSS being the sole outlier, and NL mode reveals significant model-dependent variation, particularly under adversarial conditions.

All models achieve **100% on infeasible intents**: the system robustly rejects QoS requests that exceed every CFSS pro-

file's capability envelope rather than generating an invalid intent. This is essential for safe deployment, where silently accepting an infeasible intent could later propagate into SLA violations.

*Table 2.* Success rates (%) by model and experiment. Builder C/R = Builder CFSS/RFSS; Cnst./Adv./Inf. = NL Constrained/Adversarial/Infeasible. Builder runs use structured inputs; NL runs exercise the full NL pipeline.

| Model | Bld. CFSS | Bld. RFSS | NL Cnst. | NL Adv. | NL Inf. |
|---|---|---|---|---|---|
| GPT-4.1-nano | 84 | 100 | 52 | 60 | 100 |
| GPT-4.1-mini | 100 | 100 | 100 | 90 | 100 |
| GPT-4.1 | 100 | 100 | 100 | 90 | 100 |
| GPT-5-nano | 98 | 100 | 88 | 90 | 100 |
| GPT-5-mini | 100 | 100 | 100 | 95 | 100 |
| GPT-5 | 100 | 100 | 100 | 95 | 100 |
| **Overall** | **97** | **100** | **90** | **87** | **100** |

Infeasible success = correctly rejecting QoS requests beyond all CFSS capabilities.

### 5.3. LLM Analysis: Generation and Size

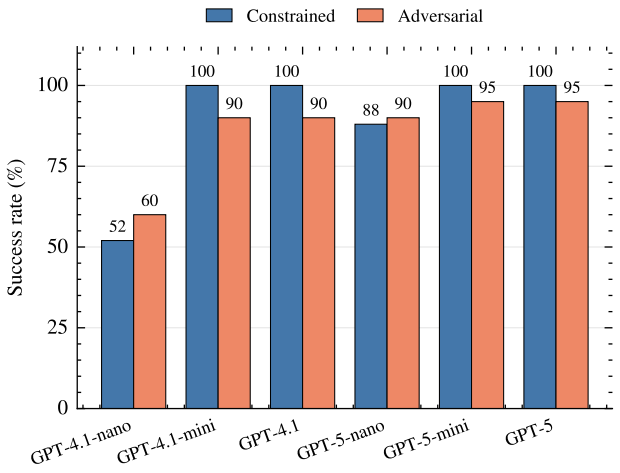


*Figure 3.* Success rates across NL constrained and adversarial experiments. Hatched bars indicate adversarial results.

Figure 3 highlights two patterns.

**Generational leap at the nano tier.** GPT-5-nano matches GPT-4.1-full on adversarial prompts (90% vs. 90%) and closes most of the constrained-prompt gap (88% vs. 100%), a substantial jump from GPT-4.1-nano's 52–60%. This suggests that next-generation mini- and nano-tier models may soon be viable candidates wherever intent processing needs to run on cost-constrained platforms, although our experiments use cloud-hosted models and do not themselves test edge or on-device deployment.

**Generation can rival size.** GPT-5-mini (95% adversarial) marginally exceeds GPT-4.1-full (90% adversarial), suggesting that within the GPT-4.1→GPT-5 transition, a generational step at the mini tier competes with the prior generation's full model on structured intent tasks. We report this as observed on a single benchmark with 120 adversarial runs per model rather than as a general law.

Per-phase analysis (Appendix C) isolates *which* stage drives these numbers. Discovery and plan decomposition are essentially solved for all models ($\geq$92%, except GPT-4.1-nano at 85–100%), whereas *constraint extraction* is the dominant bottleneck: GPT-4.1-nano reaches only 48–60% constraint precision, while GPT-5-mini reaches 95–100%. The end-to-end spread in Table 2 is almost entirely attributable to constraint-extraction error, not RDF generation or feasibility reasoning. We also observe that GPT-5 is aggressive on adversarial prompts: when a prompt mentions "ultra-low latency" next to the unmodeled "frame rate," GPT-5 sometimes drops both rather than risk an infeasible intent, trading constraint completeness for feasibility.

### 5.4. Context Grounding Reduces QoS Hallucination

A recurring failure during development was that models invented QoS constraints for metrics not present in the catalog (e.g., creating a `FrameRate` constraint for gaming profiles). We addressed this by extracting available QoS metric names from the service ontology's `req:provides` blocks and injecting them into the LLM context as "Available properties: Latency, Bandwidth, Jitter,..." with prompt rules telling the model to constrain only listed metrics.

Comparing otherwise-identical adversarial runs with and without this injection (Appendix E, Table 7), overall adversarial success improves from 61% to 87% (+26 percentage points). The effect is strongest at the mini tier, where GPT-4.1-mini rises from 50% to 90% and GPT-5-mini from 50% to 95%; larger models benefit less because they already ignore unmodeled concepts more reliably. The broader takeaway: **grounding LLM context in explicit capability metadata** yields larger reliability gains than scaling model size alone.

### 5.5. Runtime and Scalability

Since token consumption directly translates to operational cost, we compare the two modes: Builder is 40% faster (11.7 s vs. 16.4 s mean), uses 49% fewer tokens (4,582 vs. 6,848 mean), and requires one LLM call versus three. This makes Builder suitable for high-volume programmatic integration while NL mode retains flexibility for operator interaction.

Real operator catalogs are orders of magnitude larger than our benchmark; to assess scalability, we independently

benchmark the P3 matching algorithms (Appendix D, Table 6). The greedy heuristic runs 1.8× faster than the CP-SAT solver with identical requirement-satisfaction rates and a bounded cost gap, suggesting that online orchestration can rely on greedy at sub-second scale, with CP-SAT reserved for offline refinement as catalogs grow.

## 6. Related Work

**LLM-based intent management.** DMO-GPT (Mekrache et al., 2025) uses multi-agent LLMs with hierarchical planning for distributed 6G management across heterogeneous OSSs; the LLM-centric lifecycle architecture of Mekrache et al. (2024) demonstrates NL-driven decomposition, translation, and activation on a 5G testbed; and the IAN framework (Guo et al., 2025) combines RAG-enhanced LLMs with resource allocation models. All three generate intent representations from natural language but omit formal structural validation and catalog-grounded feasibility checking. LUMI (Jacobs et al., 2025) addresses trust in NL-based IBN through self-correction and verification of deployed policies, yet operates on device configurations rather than ontology-compliant RDF intents. Tageldien et al. (2025) survey LLM integration across all five IBN lifecycle phases and confirm that translation dominates current research while validation and assurance remain underexplored. Ficzere et al. (2025) reach the same conclusion from a task-classification perspective, identifying feasibility validation and structured reporting as open problems. Our system directly addresses both gaps within a single agentic pipeline.

**Semantic and ontology-based approaches.** SWIFT (Alcock et al., 2025) is the closest prior work: it employs an ontology abstracting 3GPP, ETSI NFV-MANO, and ONOS data models to translate TMF-modelled slice intents into infrastructure configurations via logical programming. However, SWIFT targets imperative configuration generation rather than validated RDF intent production, and its reasoner does not perform QoS-envelope constraint satisfaction or cross-layer decomposition. Mehmood et al. (2023) use knowledge graphs (OWL/RDF) to model intents extending the TMF Intent Common Model, demonstrating the value of semantic representations but without SHACL validation or LLM integration. Our work combines both directions: LLM-driven generation grounded in a semantic catalog with formal SHACL validation and CSP-based decomposition.

**6G architectures and standards.** The 6G-INTENSE architecture (Boutouchent et al., 2025) proposes intent-driven management at all layers—Tenant Management Platform, Domain Manager and Orchestrator, and Network-Compute Fabric—with Native AI toolkits including LLM-powered cognitive intent handlers. Our system implements the concrete NL-to-validated-intent component that such architectures require at the tenant-to-DMO interface. Brenes et al. (2025) present a unified architecture harmonizing 3GPP and TMF intent standards with LLM-based translation and closed-loop automation, validating the need for interoperable intent management but without formal feasibility reasoning. Kapoor et al. (2025) survey the broader IBN+GenAI standardization landscape across ETSI, 3GPP, TMF, and industry deployments, noting that most implementations remain at Level 2–3 automation; our catalog-grounded validation and decomposition pipeline targets the formal guarantees needed for Level 4. IETF RFC 9315 (Clemm et al., 2022) defines intent concepts; TMF ICM (TM Forum, 2024a) structures intents as RDF; ETSI ZSM (ETSI, 2019) specifies closed-loop automation. None provides formal constraint satisfaction for intent-to-resource decomposition—the gap our RequirementCapability model fills.

## 7. Conclusion

We presented an agentic LLM pipeline that goes beyond NL-to-RDF intent translation by composing three reasoning layers: catalog grounding over a semantic service specification, SHACL structural validation against the TMF Intent Ontology, and CSP-style decomposition (CFSS selection plus weighted set cover over RFSS profiles). A RequirementCapability ontology gives the three layers a shared formal substrate, so that an LLM's plausible-looking intent is only accepted when it is both well-formed and feasible against catalog capabilities. Across 930 benchmark runs spanning six GPT-4.1/5 models, the system reaches 97% success in Builder mode and 90% averaged across NL scenarios, with 100% correct rejection of infeasible requests.

Three findings stand out. First, *catalog-grounded context* reduces adversarial-prompt QoS hallucination by 26 percentage points, a larger gain than scaling model size alone. Second, *model generation can rival parameter count*: GPT-5-mini matches or slightly exceeds GPT-4.1-full on adversarial scenarios, suggesting that next-generation mini-tier models may suffice for cost-constrained intent processing. Third, *constraint extraction remains the bottleneck*: discovery, planning, and RDF generation are largely solved; the hard problem is extracting precise QoS constraints from ambiguous NL requirements.

## References

Alcock, P., Anand, R., Rotsos, C., and Race, N. SWIFT: Semantic web intent framework for intent translation. In *2025 IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE, 2025. doi: 10.1109/NOMS57970.2025.11073618.

Boutouchent, A., Mekrache, A., Ksentini, A., Adhane, G., Fonseca, J., McNamara, J., Ramantas, K.,

Palena, M., Iordache, M., Lo Cigno, R., Mostafa, S., Roy, S., and Verikoukis, C. 6G-INTENSE: Intent-driven native AI architecture supporting network-compute abstraction and sensing at the deep edge. In *Proceedings of the IEEE Vehicular Technology Conference (VTC-Spring)*. IEEE, 2025. Preprint: https://6g-intense.eu/wp-content/uploads/2025/01/Intense_VTM.pdf.

Brenes, J., Piscione, P., Kolobov, M., and Ferrigno, G. Unified intent-based management across standards: Architecture and prototype realizations. In *2025 IEEE Conference on Standards for Communications and Networking (CSCN)*. IEEE, 2025. doi: 10.1109/CSCN67557.2025.11230699.

Clemm, J., Ciavaglia, L., Granville, L., and Tantsura, J. RFC 9315: Intent-Based Networking - Concepts and Definitions. Rfc, IETF, 2022. URL https://www.rfc-editor.org/rfc/rfc9315.

ETSI. Zero-touch network and Service Management (ZSM); Reference Architecture. Etsi gs zsm 002, ETSI, 2019. URL V1.1.1.

Ficzere, D., Hollósi, G., and Varga, P. Beyond intent translation: Research gaps in the application of generative AI for intent-based networking. In *2025 IEEE Network Operations and Management Symposium (NOMS) Workshops — Workshop on Generative AI in Network Management (GAIN)*. IEEE, 2025. doi: 10.1109/NOMS57970.2025.11073701.

Guo, L., Zhang, L., Wang, J., Wu, J., Yan, Y., Sun, H., He, B., Qi, Q., and Liao, J. Intent-based autonomous network framework guided by large language model. *IEEE Transactions on Automation Science and Engineering*, 22: 22185–22197, 2025. doi: 10.1109/TASE.2025.3610906.

Jacobs, A. S., Pfitscher, R. J., Ribeiro, R. H., Granville, L. Z., Ferreira, R. A., Willinger, W., and Rao, S. G. Establishing trust for using natural language for intent-based networking. *IEEE Transactions on Network and Service Management*, 22(5):4775–4787, 2025. doi: 10.1109/TNSM.2025.3574626.

Kapoor, S., Gurbilek, G., Parra-Ullauri, J., Jangra, P. K., Khan, T. A., Duke, A., Hey, A., McHugh, D., and Corston-Petrie, A. GenAI-powered intent-based autonomous networks: Standardisation landscape and emerging industry trends. *TechRxiv*, 2025. doi: 10.36227/techrxiv.175756256.63633045. preprint.

Leivadeas, A. and Falkner, M. A survey on Intent-Based Networking. *IEEE Communications Surveys & Tutorials*, 25(1):625–655, 2023.

Martins, J., Mokrushin, L., and Orlic, M. TIO-SHACL: Comprehensive SHACL validation for TMF intent ontologies, 2026. URL https://arxiv.org/abs/2604.27359. Open-source implementation: https://github.com/EricssonResearch/tio-shacl.

Mehmood, K., Kralevska, K., and Palma, D. Knowledge-based intent modeling for next generation cellular networks. In *IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*. IEEE, 2023.

Mekrache, A., Ksentini, A., and Verikoukis, C. Intent-based management of next-generation networks: an LLM-centric approach. *IEEE Network*, 2024.

Mekrache, A., Ksentini, A., and Verikoukis, C. Dmo-gpt: An intent-driven framework for distributed 6g management and orchestration. *IEEE Communications Magazine*, 2025.

Niemöller, J., Müller, E., Maggiari, M., and Maghsoudlou, K. Evolving Service Management Toward Intent-Driven Autonomous Networks. *Ericsson Technology Review*, 2024(3):2–7, 2024. doi: 10.23919/ETR.2024.10759715.

Tageldien, M., Selim, B., and Sboui, L. Large language models in intent-based networking: A comprehensive survey across the intent lifecycle. In *2025 International Telecommunications Conference (ITC-Egypt)*. IEEE, 2025. doi: 10.1109/ITC-EGYPT66095.2025.11186656.

TM Forum. IG1253: Intent in Autonomous Networks. Introductory guide, TM Forum, 2022. URL https://www.tmforum.org/toolkits/intent/.

TM Forum. TR290: Intent Common Model. Technical report, TM Forum, 2024a. URL https://www.tmforum.org/toolkits/intent/.

TM Forum. TMF633: Service Catalog Management API v4.0, 2024b. URL https://www.tmforum.org/open-digital-architecture/open-apis/service-catalog-management-api-TMF633/v4.0.

TM Forum. TMF641: Service Ordering Management API v5.0, 2024c. URL https://www.tmforum.org/open-digital-architecture/open-apis/service-ordering-management-api-TMF641/v5.0.

## A. NL Intent Examples

Table 3 shows representative NL intents from each benchmark configuration.

*Table 3.* Example NL intents by benchmark category.

| Category | Example NL intent |
|---|---|
| Constrained | "I need a cloud gaming service with latency at most 10ms" |
| | "I need a data service with bandwidth at least 50 Mbps and availability at least 99.5 percent" |
| Adversarial | "I need a premium gaming experience with ultra-low latency and high frame rate valid during business hours with state change reporting" |
| Infeasible | "I need a video conferencing service with capacity at least 5000 participants" |

## B. RequirementCapability Ontology

Listing 2 shows a CFSS QoS capability.

*Listing 2.* Pro Gaming latency capability.

```
cfss:ProGaming req:provides [
   a req:Capability ;
   req:metric dim:Latency ;
   req:minValue "5"^^xsd:decimal ;
   req:maxValue "10"^^xsd:decimal ;
   req:unit "ms" ].
```

## C. Per-Phase Pipeline Accuracy

Tables 4 and 5 present per-phase accuracy for NL constrained (150 runs) and adversarial (120 runs) experiments. Discovery and planning are largely solved ($\geq$92%); constraint extraction is the dominant bottleneck, particularly for nano-tier models.

*Table 4.* Per-phase accuracy (%): NL constrained (150 runs).

| Model | Disc. | C.Prec | C.Rec | Prof. | RDF | Feas. |
|---|---|---|---|---|---|---|
| 4.1-nano | 100 | 48 | 52 | 52 | 48 | 52 |
| 4.1-mini | 100 | 100 | 100 | 100 | 100 | 100 |
| 4.1 | 100 | 100 | 100 | 100 | 100 | 100 |
| 5-nano | 92 | 88 | 88 | 88 | 88 | 88 |
| 5-mini | 100 | 100 | 100 | 100 | 100 | 100 |
| 5 | 100 | 100 | 100 | 100 | 100 | 100 |

## D. RFSS Matching Algorithms

Table 6 benchmarks the greedy heuristic against CP-SAT across three catalog scales (20 reps, 5 warmup). The greedy heuristic is 1.8$\times$ faster (Mann–Whitney $p < 10^{-3}$) with identical satisfaction rates; the cost gap is bounded and scale-dependent.

*Table 5.* Per-phase accuracy (%): NL adversarial (120 runs).

| Model | Disc. | C.Prec | C.Rec | Prof. | RDF | Feas. |
|---|---|---|---|---|---|---|
| 4.1-nano | 85 | 60 | 60 | 60 | 55 | 60 |
| 4.1-mini | 95 | 85 | 90 | 90 | 90 | 90 |
| 4.1 | 100 | 82 | 85 | 90 | 85 | 90 |
| 5-nano | 100 | 78 | 85 | 85 | 85 | 90 |
| 5-mini | 100 | 95 | 95 | 95 | 95 | 95 |
| 5 | 100 | 75 | 75 | 95 | 75 | 95 |

Disc.=Discovery, C.Prec/C.Rec=Constraint Precision/Recall, Prof.=Profile selection, RDF=valid RDF, Feas.=Feasibility pass.

*Table 6.* Matching algorithm performance.

| Scale | Algo | Med (ms) | Mem (KB) | Cost | Sat% |
|---|---|---|---|---|---|
| Large | Greedy | 78.8 | 230 | 1.0 | 25 |
| | CP-SAT | 121.5 | 372 | 0.4 | 25 |
| Medium | Greedy | 25.2 | 93 | 1.6 | 26 |
| | CP-SAT | 46.6 | 172 | 1.1 | 26 |
| Small | Greedy | 0.5 | 7 | 1.9 | 29 |
| | CP-SAT | 6.1 | 52 | 1.9 | 29 |

## E. Adversarial Breakdown and Context-Grounding Ablation

Table 7 isolates the effect of injecting available-property metadata. Both configurations use identical adversarial prompts (120 runs) and self-healing loops; only the prompt context differs.

*Table 7.* Adversarial success (%) without/with property injection.

| Model | Without | With | $\Delta$ |
|---|---|---|---|
| GPT-4.1-nano | 40 | 60 | +20 |
| GPT-4.1-mini | 50 | 90 | +40 |
| GPT-4.1 | 75 | 90 | +15 |
| GPT-5-nano | 70 | 90 | +20 |
| GPT-5-mini | 50 | 95 | +45 |
| GPT-5 | 80 | 95 | +15 |
| **Overall** | **61** | **87** | **+26** |

Mini-tier models gain most (+40/+45); full-size models gain less (+15) as they already suppress unmodeled concepts.

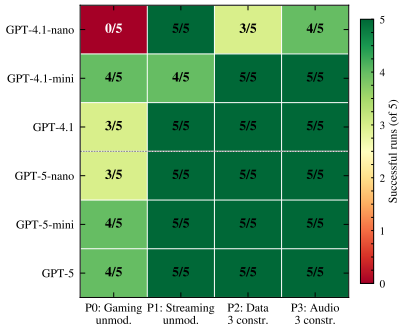


*Figure 4.* Per-prompt adversarial success. "unmod." = unmodeled metric. P2/P3 use only modeled metrics.