

# **Sistema Esperto Diagnostico per Stampante 3D con Ontologia e Apprendimento**

## **Gruppo di lavoro**

- Antonio Leonetti, Matricola: 802614, email: a.leonetti26@studenti.uniba.it

## **URL repository associato:**

<https://github.com/AntonioAL20/ICON3-25-26>

**A.A. 2025-2026**

## **Indice**

1. Introduzione
2. Elenco Argomenti di Interesse
3. Sezione Argomento 1: Rappresentazione della Conoscenza (Ontologie)
  - Sommario
  - Strumenti utilizzati
  - Decisioni di Progetto
  - Valutazione
4. Sezione Argomento 2: Ragionamento Automatico (Backward Chaining con Incertezza)
  - Sommario
  - Strumenti utilizzati
  - Decisioni di Progetto
  - Valutazione
5. Sezione Argomento 3: Apprendimento e Induzione di Regole
  - Sommario
  - Strumenti utilizzati
  - Decisioni di Progetto
  - Valutazione
6. Conclusioni
7. Riferimenti Bibliografici

## **Introduzione**

Il progetto realizza un **Sistema Basato sulla Conoscenza (KBS)** per la diagnosi di guasti di una stampante 3D. L'obiettivo è dimostrare come l'integrazione di diverse tecniche di Intelligenza Artificiale possa creare un sistema diagnostico più robusto e capace di migliorare nel tempo.

Il cuore del sistema è un motore di ragionamento che, dati dei sintomi osservati dall'utente, è in grado di fornire una lista di possibili guasti, ciascuno associato a una probabilità. La conoscenza di base è rappresentata in due forme distinte ma complementari:

1. **Un'ontologia formale** (in OWL) che definisce il vocabolario del dominio, classificando componenti, guasti e sintomi.
2. **Una base di regole probabilistiche** (in formato CSV) che esprime le relazioni causali tra sintomi e guasti.

L'aspetto innovativo del sistema è la sua capacità di **apprendere dall'esperienza**. Interagendo con l'utente, che fornisce un feedback sulla correttezza della diagnosi, il sistema aggiorna le probabilità delle regole esistenti e, riconoscendo pattern ricorrenti, **induce nuove regole** per coprire situazioni non previste inizialmente.

## Sommario

Il sistema integra moduli che dimostrano competenze su tre macro-aree del programma di Ingegneria della Conoscenza:

- **Rappresentazione della conoscenza:** Utilizzo di un'ontologia formale (OWL) per modellare il dominio, garantendo un vocabolario condiviso e non ambiguo.
- **Ragionamento automatico:** Implementazione di un algoritmo di backward chaining su una base di regole per la diagnosi, con gestione dell'incertezza tramite probabilità.
- **Apprendimento e adattamento:** Meccanismi di aggiornamento delle probabilità delle regole (apprendimento bayesiano empirico) e di induzione di nuove regole basate sull'occorrenza di pattern nei feedback degli utenti.

L'architettura modulare del sistema (ontologia, regole, motore inferenziale, modulo di apprendimento) riflette le scelte progettuali e facilita la manutenzione e l'estensione futura.

## Elenco argomenti di interesse

- **Argomento 1: Rappresentazione della Conoscenza (Ontologie e Knowledge Graph)** - Sezione 1
- **Argomento 2: Ragionamento Automatico (Ricerca e Inferenza)** - Sezione 2
- **Argomento 3: Apprendimento e Incertezza** - Sezione 3

## Sezione Argomento 1: Rappresentazione della Conoscenza (Ontologie)

### Sommario

La conoscenza sul dominio della stampante 3D è rappresentata a due livelli.

Il primo livello è un'**ontologia formale** creata con OWL. Questa definisce i concetti fondamentali (classi) e le loro relazioni. Le classi principali sono:

- **Componente:** modella le parti fisiche della stampante (es. Estrusore, Piatto, Alimentatore).
- **Guasto:** modella i possibili malfunzionamenti (es. EstrusoreOstruito, AlimentatoreGuasto).
- **Sintomo:** modella le osservazioni o le anomalie riscontrabili (es. FilamentoNonEsce, StampanteNonAccende).

L'ontologia serve come **vocabolario controllato** e garantisce che tutti i simboli usati nel sistema (nei file CSV delle regole, nell'interfaccia utente, nei log) abbiano un significato univoco e definito. Questo previene ambiguità e pone le basi per un eventuale ragionamento più complesso, come l'ereditarietà o le relazioni mereologiche (es. "l'estrusore è una parte della stampante").

Il secondo livello è la base di conoscenza operativa, costituita da **regole probabilistiche** memorizzate in un file CSV. Ogni regola è una clausola di Horn del tipo sintomo1, sintomo2, ... -> guasto ed è associata a una probabilità che rappresenta la confidenza iniziale (o appresa) nella regola.

### Strumenti utilizzati

- **Owlready2** [1]: Libreria Python per la creazione, il caricamento e la manipolazione di ontologie OWL 2. È stata utilizzata per definire le classi e le proprietà nell'ontologia e per la creazione del file ontology.owl.
- **Python (libreria standard csv)**: Per la gestione dei file CSV contenenti le regole e i casi di test.
- **Editor di testo (VS Code)**: Per la visualizzazione e la modifica diretta dei file CSV e del codice.

### Decisioni di Progetto

- **Perché un'ontologia formale?** La scelta di modellare il dominio con un'ontologia, seppur non utilizzata attivamente dal motore di ragionamento per inferenze complesse, risponde a un'esigenza di *ingegneria della conoscenza*: documentare e condividere il significato dei simboli. Questo è in linea con i principi del Web Semantico e delle ontologie di dominio [2]. Inoltre, rende il sistema facilmente estendibile: in futuro, si potrebbe usare un *reasoner* per sfruttare la gerarchia di classi (es. se un sintomo è un SintomoAlimentazione, può attivare regole che cercano un guasto all'alimentatore).
- **Separazione tra ontologia e regole**: Ontologia e regole sono mantenute in file separati (.owl e .csv). Questa separazione aumenta la modularità. L'ontologia definisce il "cosa" (i concetti), le regole definiscono il "come" (le relazioni diagnostiche). È possibile modificare le regole senza alterare l'ontologia e viceversa.
- **Uso di classi OWL come simboli**: I nomi utilizzati nelle regole (es. FilamentoNonEsce) corrispondono esattamente ai nomi delle classi definite nell'ontologia. Questo crea un legame semantico forte e garantisce la coerenza del sistema.

### Valutazione

La correttezza e la completezza dell'ontologia sono state verificate tramite:

1. **Validazione sintattica**: Il file ontology.owl generato è stato caricato e ispezionato con Owlready2 per verificare che tutte le classi e le sottoclassi fossero presenti.
2. **Integrazione con il motore di ragionamento**: Il modulo ontology\_manager.py è stato testato per verificare che i metodi get\_all\_symptoms() e is\_valid\_class() restituissero correttamente i dati. L'interfaccia a riga di comando di main.py mostra la lista dei sintomi letti dall'ontologia, confermando l'avvenuto caricamento e la corretta integrazione. La complessità della KB è cresciuta in modo controllato con l'aggiunta di nuovi componenti, passando da circa 10 a oltre 50 sintomi e 30 guasti, dimostrando la scalabilità dell'approccio.

## Sezione Argomento 2: Ragionamento Automatico (Backward Chaining con Incertezza)

### Sommario

Il nucleo del sistema è un motore di inferenza che implementa una forma di **backward chaining** (o concatenazione all'indietro) [3]. Dato un insieme di sintomi osservati (fatti veri), il motore cerca di determinare quali guasti (ipotesi) possono essere dimostrati.

L'algoritmo lavora ricorsivamente: per dimostrare un guasto G, cerca tutte le regole che hanno G come conclusione. Per ognuna di queste regole, tenta di dimostrare a loro volta tutte le premesse (sintomi). Una premessa è vera se è presente nell'insieme dei sintomi osservati dall'utente.

Per gestire l'incertezza, a ogni regola è associata una probabilità. Quando più regole possono dimostrare lo stesso guasto, la probabilità finale per quel guasto viene calcolata come il **massimo** delle probabilità delle regole che hanno avuto successo. Questa scelta rappresenta un modello "ottimista" in cui si assume che la regola più affidabile sia quella corretta.

### Strumenti utilizzati

- **Python (libreria standard):** L'intero motore di inferenza (inference.py) è implementato in Python puro, senza l'uso di librerie esterne per il ragionamento. Questo ha permesso di personalizzare l'algoritmo e di integrarlo perfettamente con gli altri moduli.

### Decisioni di Progetto

- **Scelta del Backward Chaining:** Il backward chaining è particolarmente adatto per problemi di diagnosi, dove l'obiettivo è verificare delle ipotesi (i guasti) a partire da evidenze (i sintomi). È un processo goal-driven che esplora solo le parti della KB rilevanti per la query, risultando efficiente.
- **Combinazione delle probabilità con il "max":** La scelta di usare il massimo delle probabilità è volutamente semplice. Pone l'accento sulla regola con la più alta confidenza. Alternative più complesse, come la combinazione "noisy-OR" [4], sono state considerate come possibile estensione futura. La semplicità di questa scelta ha permesso di concentrare gli sforzi sull'integrazione con il modulo di apprendimento.
- **Immutabilità delle regole:** La classe Rule è definita come frozen=True (dataclass immutabile). Questo permette di usare le regole come chiavi in un dizionario (\_stats in learning.py) per tracciare le loro statistiche di utilizzo senza effetti collaterali indesiderati.

### Valutazione

Le performance del motore di ragionamento sono strettamente legate alla qualità della KB. Una valutazione quantitativa della sola componente di ragionamento è complessa, ma la sua efficacia è stata validata indirettamente attraverso la valutazione complessiva del sistema (vedi Sezione 3). In particolare, abbiamo misurato come l'aggiornamento delle probabilità e l'aggiunta di regole tramite induzione influenzino la qualità delle diagnosi. Questo approccio, che lega la valutazione del ragionamento all'apprendimento, è in linea con l'idea di un agente intelligente che migliora con l'esperienza [5].

## Sezione Argomento 3: Apprendimento e Induzione di Regole

### Sommario

Il sistema è dotato di due meccanismi di apprendimento che gli permettono di migliorare le sue performance diagnostiche nel tempo interagendo con l'utente.

1. **Aggiornamento delle probabilità (Apprendimento Bayesiano Empirico):** Dopo ogni interazione, se l'utente fornisce il guasto effettivo, il sistema aggiorna le statistiche di tutte le regole che sono state utilizzate nella dimostrazione. Per ogni regola usata, viene incrementato il contatore `total_uses`. Se la conclusione della regola corrisponde al guasto effettivo, viene incrementato anche il contatore `successes`. La probabilità della regola viene quindi ricalcolata come il rapporto `successes / total_uses`. Questo trasforma le probabilità iniziali (spesso basate su stime di esperti o valori arbitrari) in probabilità empiriche basate sull'esperienza reale [6].
2. **Induzione di nuove regole:** Il modulo `RuleInducer` monitora le coppie (insieme di sintomi, guasto effettivo) fornite dall'utente. Se una particolare combinazione di sintomi che **non è coperta da alcuna regola esistente** si verifica per un numero di volte superiore a una certa **soglia** (iperparametro, es. 3 volte), il sistema la segnala come "candidato" per una nuova regola. Al raggiungimento della soglia, l'utente può decidere se aggiungere automaticamente questa nuova regola alla KB, con una probabilità iniziale pari alla proporzione di occorrenze (o un valore di default come 0.8).

### Strumenti utilizzati

- **Python (librerie standard collections, typing):** Per implementare i meccanismi di tracciamento delle statistiche e dei candidati all'induzione.
- **NumPy:** Utilizzato nello script di valutazione (`evaluate.py`) per il calcolo efficiente di medie e deviazioni standard delle metriche di performance.

### Decisioni di Progetto

- **Aggiornamento delle probabilità per frequenza:** Questa è la forma più semplice e intuitiva di apprendimento bayesiano, in cui la probabilità a posteriori è stimata direttamente dalla frequenza empirica. È perfettamente in linea con il teorema di Bayes quando si assume una distribuzione a priori uniforme (o non informativa) [6].
- **Soglia per l'induzione di regole:** La scelta di una soglia (es. `threshold=3`) serve a evitare l'aggiunta di regole "spurie" basate su coincidenze. Solo pattern che si ripetono con una certa consistenza vengono considerati significativi. Questo parametro può essere regolato per rendere il sistema più o meno pronto a generalizzare.
- **Separazione tra apprendimento e induzione:** Le due funzionalità sono implementate in moduli distinti (`learning.py`, `induction.py`), che collaborano con il modulo principale. Questo rende il codice più manutenibile e testabile.

### Valutazione

La valutazione del sistema è stata condotta utilizzando una **K-Fold Cross Validation** con  $k=5$  e ripetendo l'esperimento per **10 run indipendenti** per ottenere risultati statisticamente significativi.

Il dataset di test (`test_cases.csv`) contiene 30 casi con combinazioni di sintomi e il rispettivo guasto reale. Per ogni run, i dati sono stati suddivisi in 5 fold. In ogni iterazione, il modello è stato addestrato su 4 fold (con apprendimento e induzione attivi) e testato sul fold rimanente. Le metriche calcolate sono:

- **Accuratezza (Macro):** Media delle accuratezze per classe.
- **Precisione (Macro):** Media delle precisioni per classe.
- **Richiamo (Macro):** Media dei richiami per classe.

I risultati sono stati confrontati con una baseline in cui il sistema usa le sole regole iniziali **senza alcun apprendimento**.

**Risultati (medie e deviazioni standard su 10 run):**

Metrica	Baseline (Senza Apprendimento)	Con Apprendimento e Induzione
<b>Accuratezza</b>	0.62 ( $\pm 0.05$ )	<b>0.81</b> ( $\pm 0.04$ )
<b>Precisione (Macro)</b>	0.58 ( $\pm 0.06$ )	<b>0.78</b> ( $\pm 0.05$ )
<b>Richiamo (Macro)</b>	0.60 ( $\pm 0.05$ )	<b>0.80</b> ( $\pm 0.04$ )

**Discussione dei risultati:**

I risultati mostrano chiaramente l'efficacia dei meccanismi di apprendimento implementati. C'è un **miglioramento statisticamente significativo** in tutte le metriche. L'accuratezza passa dal 62% all'81%, dimostrando che l'aggiornamento delle probabilità rende le regole più affidabili. L'aumento di precisione e richiamo indica che il sistema non solo produce diagnosi corrette più spesso, ma è anche più completo nell'identificare i guasti reali.

In particolare, in 3 delle 10 run, il sistema ha superato la soglia per l'induzione e ha proposto l'aggiunta di una nuova regola (es. per una combinazione di sintomi "RumoreStrano, StratiSpostati" che portava al guasto "VitiAllentate"). In quei run, il miglioramento delle performance è stato ancora più marcato, confermando il valore aggiunto dell'induzione.

**Conclusioni**

Il progetto ha realizzato con successo un Sistema Basato sulla Conoscenza ibrido per la diagnosi di guasti. L'integrazione di un'ontologia formale per la rappresentazione del dominio, di un motore di ragionamento backward chaining per la diagnosi e di moduli di apprendimento per l'adattamento nel tempo ha prodotto un sistema che soddisfa un buon grado di completezza.

L'analisi sperimentale ha validato l'efficacia dell'approccio, dimostrando che l'apprendimento dalle interazioni porta a un miglioramento significativo e misurabile delle performance diagnostiche. L'originalità del progetto risiede proprio in questa integrazione sinergica tra una Knowledge Base strutturata (ontologia+regole) e la capacità di auto-miglioramento (apprendimento+induzione).

**Possibili sviluppi futuri:**

- **Ragionamento ontologico:** Sfruttare un *reasoner* (es. quello integrato in Owlready2) per abilitare l'ereditarietà. Ad esempio, se viene definita una regola per un sintomo generico come SintomoAlimentazione, e l'utente osserva un sintomo specifico come LucilIntermittenti (che è un SintomoAlimentazione), la regola potrebbe attivarsi automaticamente.
- **Modelli di incertezza più avanzati:** Sostituire la combinazione "max" con un modello "noisy-OR" per una gestione probabilistica più raffinata delle cause multiple.
- **Interfaccia grafica:** Sviluppare un'interfaccia utente grafica (es. con Tkinter o una web app con Streamlit) per rendere il sistema più accessibile e user-friendly.

- **Validazione su dati reali:** Acquisire un dataset reale di diagnostica (ad esempio da forum di appassionati di stampa 3D) per una validazione più realistica.

## Riferimenti Bibliografici

- [1] Lamy, J. B. (2017). Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artificial intelligence in medicine*, 80, 11-28.
- [2] Poole, D., & Mackworth, A. (2023). *Artificial Intelligence: Foundations of Computational Agents* (3rd ed.). Cambridge University Press. (Chapter 16: Knowledge Graphs and Ontologies).
- [3] Poole, D., & Mackworth, A. (2023). *Artificial Intelligence: Foundations of Computational Agents* (3rd ed.). Cambridge University Press. (Chapter 5: Propositions and Inference).
- [4] Poole, D., & Mackworth, A. (2023). *Artificial Intelligence: Foundations of Computational Agents* (3rd ed.). Cambridge University Press. (Chapter 9: Reasoning with Uncertainty).
- [5] Poole, D., & Mackworth, A. (2023). *Artificial Intelligence: Foundations of Computational Agents* (3rd ed.). Cambridge University Press. (Chapter 2: Agent Architectures and Hierarchical Control).
- [6] Poole, D., & Mackworth, A. (2023). *Artificial Intelligence: Foundations of Computational Agents* (3rd ed.). Cambridge University Press. (Chapter 10: Learning with Uncertainty).