**Astraal LXP**

**Personalized Content Sequencing Engine (PCSE)**

*(Standalone, ML-driven sequencing layer)*

**1️⃣ Problem (Clear & Precise)**

In most LMS/LXP systems:

* Content is linear
* Every learner sees the same sequence
* Modules are fixed
* Pace is assumed uniform

But learners differ in:

* Prior knowledge
* Speed of comprehension
* Retention patterns
* Topic familiarity
* Fatigue & cognitive load

Result:

* Advanced learners feel slowed down
* Beginners feel overwhelmed
* Sequencing becomes inefficient

**2️⃣ Solution (Standalone Framing)**

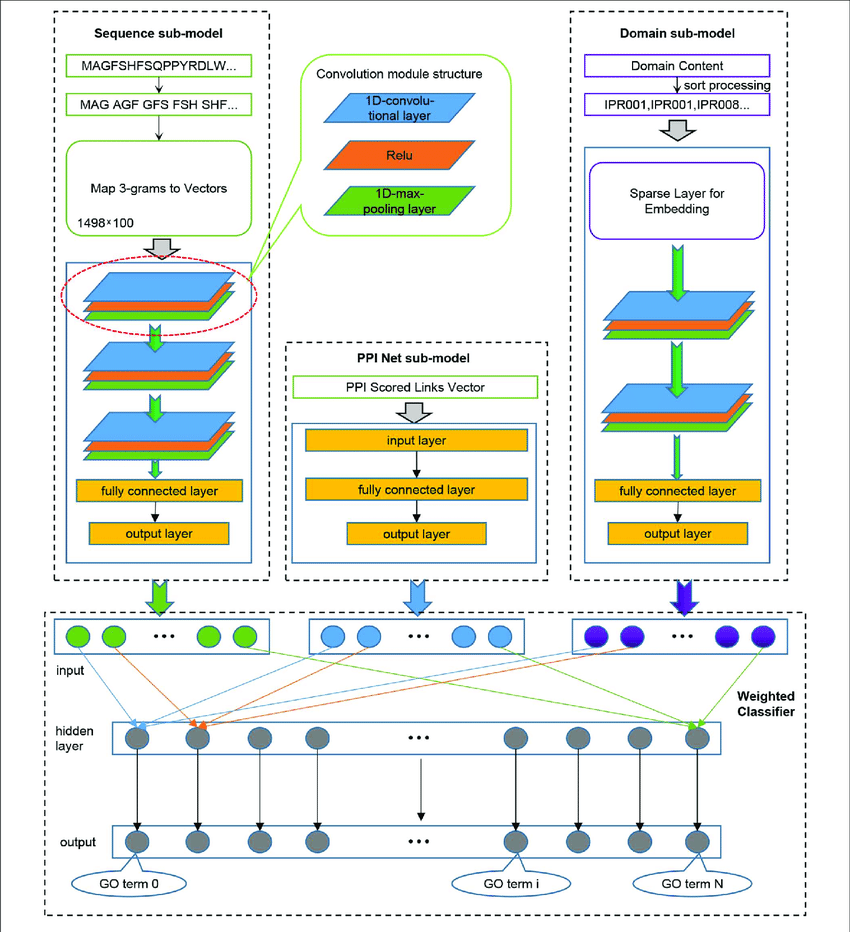
Build a **Personalized Content Sequencing Engine (PCSE)** that:

* Reorders learning units within a course
* Uses ML ranking to prioritize next content
* Accounts for readiness & pace
* Preserves full learner visibility
* Allows override

This is sequencing, not adaptation of difficulty.

**3️⃣ Architectural Overview (UWAMP + GoDaddy Compatible)**







**System Flow**

Learner Activity (PHP UI)

↓

Learning Events Store (MySQL)

↓

Sequencing Feature Builder (Python CLI)

↓

ML Ranking Model

↓

Ordered Content List (MySQL)

↓

PHP Rendering Layer

Constraints:

* No long-running Python servers
* CLI execution only
* PHP triggers model
* Shared-hosting compliant

**4️⃣ Core Data Model**

**4.1 Content Units**

CREATE TABLE content\_units (

unit\_id INT AUTO\_INCREMENT PRIMARY KEY,

course\_id INT,

unit\_title VARCHAR(255),

difficulty\_level FLOAT,

prerequisite\_unit INT NULL,

estimated\_time INT

);

**4.2 Learner Content Activity**

CREATE TABLE learner\_content\_activity (

learner\_id INT,

unit\_id INT,

completion\_status BOOLEAN,

time\_spent INT,

assessment\_score FLOAT,

last\_access DATETIME

);

**4.3 Sequenced Output**

CREATE TABLE personalized\_sequence (

learner\_id INT,

course\_id INT,

unit\_id INT,

rank\_position INT,

calculated\_on DATETIME

);

**5️⃣ Sequencing Intelligence Logic**

We use ML ranking.

For each unit, compute:

**Feature Vector**

[

prior\_score,

time\_efficiency\_ratio,

difficulty\_gap,

prerequisite\_completed,

recency\_of\_activity

]

**Feature Explanation**

| **Feature** | **Meaning** |
| --- | --- |
| prior\_score | Performance on related units |
| time\_efficiency\_ratio | Actual time vs expected time |
| difficulty\_gap | Difference between readiness & unit difficulty |
| prerequisite\_completed | Boolean 0/1 |
| recency | How recently learner engaged |

**6️⃣ ML Model Choice**

Because you are on shared hosting:

Recommended:

* Logistic Regression
* Light Gradient Boosted Trees
* XGBoost (optional offline training)

Training happens offline.  
Inference runs via CLI.

**7️⃣ Python Sequencing Engine**

**7.1 Model Training (Offline)**

from sklearn.linear\_model import LogisticRegression

import pickle

model = LogisticRegression()

model.fit(X\_train, y\_train)

pickle.dump(model, open("sequencer.pkl","wb"))

**7.2 Sequencing Script (CLI)**

content\_sequencer.py

import sys

import mysql.connector

import pickle

import pandas as pd

learner\_id = sys.argv[1]

course\_id = sys.argv[2]

model = pickle.load(open("sequencer.pkl","rb"))

db = mysql.connector.connect(...)

cursor = db.cursor(dictionary=True)

# Fetch candidate units

cursor.execute("""

SELECT \* FROM content\_units

WHERE course\_id=%s

""",(course\_id,))

units = cursor.fetchall()

feature\_rows = []

for unit in units:

# Fetch learner activity

cursor.execute("""

SELECT \* FROM learner\_content\_activity

WHERE learner\_id=%s AND unit\_id=%s

""",(learner\_id,unit['unit\_id']))

activity = cursor.fetchone()

prior\_score = activity['assessment\_score'] if activity else 0

time\_ratio = (activity['time\_spent']/unit['estimated\_time']) if activity else 1

difficulty\_gap = abs(unit['difficulty\_level'] - prior\_score)

prerequisite\_done = 1

feature\_rows.append([

prior\_score,

time\_ratio,

difficulty\_gap,

prerequisite\_done

])

df = pd.DataFrame(feature\_rows)

scores = model.predict\_proba(df)[:,1]

ranked = sorted(zip(units, scores), key=lambda x: x[1], reverse=True)

# Store ranking

cursor.execute("DELETE FROM personalized\_sequence WHERE learner\_id=%s AND course\_id=%s",(learner\_id,course\_id))

for i, item in enumerate(ranked):

cursor.execute("""

INSERT INTO personalized\_sequence

VALUES (%s,%s,%s,%s,NOW())

""",(learner\_id,course\_id,item[0]['unit\_id'],i+1))

db.commit()

**8️⃣ PHP Trigger**

exec("python3 /home/user/python/content\_sequencer.py $learner\_id $course\_id");

**9️⃣ PHP Rendering Layer**

Instead of:

SELECT \* FROM content\_units ORDER BY default\_sequence;

Use:

SELECT cu.\*

FROM personalized\_sequence ps

JOIN content\_units cu ON ps.unit\_id=cu.unit\_id

WHERE ps.learner\_id=? AND ps.course\_id=?

ORDER BY ps.rank\_position;

**🔟 Governance & Transparency**

Important:

* Learners see: “Content order optimized for your learning pace.”
* Learners can switch to default order.
* Faculty can toggle personalized sequencing.
* No unit is removed.
* No hidden skipping.

Microcopy:

The order of topics may adjust to better align with your learning pace. You can return to the standard sequence at any time.

**11️⃣ System Architecture Summary**

Content Units (MySQL)

↓

Learner Activity Logs

↓

Feature Builder (Python)

↓

ML Ranking

↓

Personalized Order Table

↓

PHP UI Rendering

Fully compliant with:

* UWAMP
* GoDaddy Shared Hosting
* CLI-based execution

**12️⃣ What This Enables**

* Faster progression for advanced learners
* Reinforcement-first sequencing for beginners
* Reduced cognitive overload
* Higher retention
* More efficient content consumption

Without:

* Changing course structure
* Predicting outcomes
* Altering certification logic

**PART 1️⃣**

**Integrating Content Sequencing with Engagement Scoring**

We are not merging modules.  
We are creating **signal sharing with strict boundaries**.

**1.1 Design Principle**

Engagement should:

* Inform sequencing
* Not dominate sequencing
* Never penalize
* Never label learners

Sequencing remains content-order optimization.  
Engagement becomes a pacing and attention signal.

**1.2 New Feature Additions to Sequencing Model**

From Engagement Engine, we import:

| **Engagement Signal** | **Use in Sequencing** |
| --- | --- |
| Continuity score | Adjust pacing intensity |
| Interaction score | Increase complexity tolerance |
| Participation | Delay heavy collaborative content if low |
| Reflection | Insert recap units earlier |

**1.3 Database Integration**

Add a lightweight integration view:

CREATE VIEW learner\_engagement\_summary AS

SELECT learner\_id,

overall\_engagement,

continuity\_score,

interaction\_score

FROM engagement\_scores;

Sequencing engine queries this view.

**1.4 Modified Feature Vector**

Updated sequencing features:

[

prior\_score,

time\_efficiency\_ratio,

difficulty\_gap,

prerequisite\_completed,

engagement\_overall,

continuity\_score

]

**1.5 Updated Python Snippet**

Inside content\_sequencer.py:

cursor.execute("""

SELECT overall\_engagement, continuity\_score

FROM learner\_engagement\_summary

WHERE learner\_id=%s

""",(learner\_id,))

eng = cursor.fetchone()

engagement\_level = eng['overall\_engagement'] if eng else 0.5

continuity = eng['continuity\_score'] if eng else 0.5

Then add these to feature row.

This allows sequencing to adapt to:

* Cognitive fatigue
* Inconsistent engagement
* High momentum states

Still no penalties.

**PART 2️⃣**

**Reinforcement Learning Upgrade Path**

This is optional, future-ready.

Important:  
GoDaddy shared hosting cannot support real-time RL agents.

So we design:

Offline batch reinforcement updates.

**2.1 Why Reinforcement Learning?**

Current system:

* Predicts next best unit based on static model.

Reinforcement Learning:

* Learns from sequencing outcomes over time.
* Optimizes long-term engagement & completion.

**2.2 Framing as a Multi-Armed Bandit Problem**

Each content unit = an “arm”  
Reward = engagement improvement + completion success

We implement:

* ε-greedy strategy
* Thompson sampling (optional upgrade)

**2.3 New Table for Reinforcement Tracking**

CREATE TABLE sequencing\_rewards (

learner\_id INT,

unit\_id INT,

reward\_value FLOAT,

recorded\_on DATETIME

);

Reward formula example:

reward =

(completion\_status \* 0.5) +

(engagement\_delta \* 0.5)

**2.4 Batch RL Update Script**

rl\_update.py

import mysql.connector

# Simplified epsilon-greedy adjustment

epsilon = 0.1

# Update content ranking weights based on reward history

This runs weekly via cron.

It adjusts:

* Weight multipliers for content difficulty
* Engagement influence factor
* Reinforcement score per unit

**2.5 Safety Constraints**

RL must:

* Never remove mandatory units
* Never reorder prerequisite logic
* Never override faculty lock
* Log every update

This ensures governance.

**PART 3️⃣**

**Faculty Controls UI Design**

This is critical.

Without governance, sequencing becomes opaque.

**3.1 Faculty Control Dashboard Structure**

**Page: Content Sequencing Governance**

**Section A: Mode Selection**

* ☑ Standard Sequence (Default Order)
* ☑ Personalized Sequencing
* ☑ Personalized + Engagement-Aware
* ☑ Reinforcement Enhanced (Experimental)

Faculty chooses course-level mode.

**Section B: Lock Critical Units**

Faculty can:

* Mark unit as “Fixed Position”
* Mark unit as “Must Appear Before X”
* Lock prerequisite chain

UI example:

[✓] Lock Unit Order

[✓] Enforce Prerequisites

[ ] Allow Difficulty Reordering

**Section C: Sequencing Sensitivity Controls**

Sliders:

* Engagement Influence (Low / Medium / High)
* Pace Sensitivity
* Difficulty Adjustment Strength

These map to ML weight multipliers.

**Section D: Audit Log**

Table:

| **Date** | **Learner** | **Old Position** | **New Position** | **Reason** |
| --- | --- | --- | --- | --- |

This builds trust.

**3.2 Database for Faculty Controls**

CREATE TABLE sequencing\_settings (

course\_id INT,

mode VARCHAR(50),

engagement\_weight FLOAT,

rl\_enabled BOOLEAN,

last\_updated DATETIME

);

**3.3 PHP Faculty Panel Example**

<form method="post">

<select name="mode">

<option>Standard</option>

<option>Personalized</option>

<option>Engagement-Aware</option>

</select>

<input type="range" name="engagement\_weight" min="0" max="1" step="0.1">

<button>Save Settings</button>

</form>

**Full Integrated Architecture**

Learner Activity

↓

Engagement Engine

↓

Sequencing Engine

↓

RL Batch Optimizer

↓

Faculty Governance Layer

↓

Personalized Order

↓

Learner UI

All modules remain:

* CLI Python
* MySQL-driven
* PHP-triggered
* Shared-hosting compliant

**Strategic Impact**

You now have:

* Engagement-aware sequencing
* Self-optimizing order logic
* Faculty-controlled intelligence
* Explainable AI path
* Scalable upgrade path

This moves Astraal toward:

Structured Intelligence  
→ Behavioral Intelligence  
→ Optimization Intelligence