Total development plan (MVP → pilot) + LLM request flow

Phase 0 — Research-ready MVP scope (fastest publishable)

1. **LTI 1.3 Tool Provider** (login/launch + role handling)
2. **Canvas adapter** (fetch assignment info + rubric + due/submitted timestamps + submission file)
3. **Admin Policy Pack** (uploaded once per institution; always available)
4. **Evaluation packet builder**
5. **LLM call** (upload files + structured prompt)
6. **Post single comment** back to submission
7. **Audit log** (IDs + timestamps + prompt version; no file storage)

Phase 1 — Reliability upgrades

- Week slide selection (instructor chooses module files)
- Robust file extraction fallback (if file upload not supported for some formats)
- Prompt versioning + regression test set
- “Unreadable evidence” detection (tiny images, scanned pages)

Phase 2 — Study + deployment

- Instructor review UI (approve/edit before posting)
- Rubric-missing “overall mode”
- Analytics dashboard (latency, acceptance rate)

Implementation plan (Python)

Recommended stack

- **Backend:** Django + Django REST Framework
- **Async jobs:** Celery + Redis (submission polling + evaluation queue)

- **DB:** Postgres (tenant config + minimal logs)
- **Deployment:** Docker + HTTPS (required for LTI)

Core services/modules

1. **lti/**
    - OIDC login initiation endpoint
    - Launch endpoint (JWT verify, role extract, context store)
2. **canvas/**
    - Canvas API client (token exchange if needed; fetch course/assignment/submission/rubric)
3. **policy/**
    - Tenant policy pack upload + parsing (admin)
4. **packet/**
    - build_evaluation_packet(context)
5. **llm/**
    - upload files + send response request + delete uploads
6. **feedback/**
    - format_comment() + post_comment_to_lms()
7. **jobs/**
    - polling job + evaluation job + retry policy

Minimal endpoints (MVP)

- `POST /lti/login`
- `POST /lti/launch`
- `POST /admin/policy-pack` (upload policy PDF)
- `POST /evaluate/submission/{submission_id}` (manual run for instructor)
- Background job: `poll_new_submissions(course_id, assignment_id)`

LLM request pattern (no vendor name in UI text)

File inputs guidance (official OpenAI docs):

- Upload file via Files/Uploads API, then reference using `input_file` in the Responses API. (OpenAI Platform)

Python pseudo-implementation (server-side)

from openai import OpenAI

```
client = OpenAI()

def run_pre_eval(policy_file_id, slides_file_id, submission_file_id, prompt_text):
    # Build input array (policy always present; slides optional)
    inputs = [
        {"type": "input_file", "file_id": policy_file_id},
    ]
    if slides_file_id:
        inputs.append({"type": "input_file", "file_id": slides_file_id})
    inputs.append({"type": "input_file", "file_id": submission_file_id})
    inputs.append({"type": "input_text", "text": prompt_text})

    resp = client.responses.create(
        model="gpt-4.1-mini",  # choose your model
        input=inputs,
    )
    return resp.output_text  # then post to LMS
```

File upload (PDF recommended for “file inputs”)

OpenAI’s “file inputs” guide shows uploading a PDF first, then referencing by file_id. (OpenAI Platform)
If DOCX/PPTX support is inconsistent, the safe MVP is:

- Convert DOCX/PPTX → PDF (server-side) OR extract text and send as `input_text`.

Prompt template (structured + rubric grounded, comment-only)

Your backend should generate something like:

- “You are an AI Teaching Assistant generating a **pre-evaluation** comment. Instructor final.”
- Provide: rubric criteria list + points, assignment instructions, due/submitted timestamps, computed late tier, policy excerpt summary.
- Output format exactly:
    1. Overall short comment
    2. Criterion lines with `Name — x/y` + `Comment:`
    3. Final `Possible Final Grade: X/Y`
    4. “Source notice” if slides missing; “Rubric notice” if rubric missing.

Development plan (Python MVP) — APIs + LLM request

A) Services (minimal, realistic)

- **Backend:** Django + Django REST Framework
- **Async:** Celery + Redis (submission polling + evaluation jobs)
- **DB:** Postgres (store only IDs, configs, logs, comment text)
- **Storage:** none for files (stream/temporary only), then delete
- **LLM integration:** "upload files → evaluate → delete files" pattern

B) Core modules

1. **LTI 1.3 Module**
    - OIDC login initiation endpoint
    - LTI launch endpoint (verify JWT id_token)
    - Store tenant: `issuer`, `client_id`, `deployment_id`
    - Placement enable/disable per assignment (Deep Linking optional for MVP)
2. **Canvas Connector (MVP)**
    - Fetch assignment instructions + due datetime (Assignments API) (Canvas)
    - Fetch rubric details (Rubrics API) (Instructure Developer Documentation)
    - Fetch submissions + attachments (Submissions API) (Instructure Developer Documentation)
    - Post a submission comment (via Submissions "grade/comment" endpoint in docs; Canvas points to this under Submissions API usage) (Instructure Developer Documentation)
    - Deep Linking support (clean instructor UX) (Instructure Developer Documentation)
3. **Policy Engine (deterministic)**
    - Admin uploads **University Policy Pack** once at tenant setup
    - Parse late tiers into structured rules (recommended)
    - Compute: `late_minutes`, `late_tier`, `deduction_percent`
    - Never penalize for missing slides (only add notice)
4. **Packet Builder**
    - Creates the "Source Pack":
        - assignment instructions

        - rubric (or “rubric missing” flag)
        - due/submitted timestamps + computed late tier
        - policy pack (always)
        - week slides (optional)
        - submission file (always)
5. **LLM Orchestrator**
    - Build a **single strict prompt**:
        - “Generate ONE consolidated comment”
        - “If rubric missing: overall evaluation mode”
        - “If slides missing: add source-availability notice, don’t penalize”
        - “Use evidence pointers (slide/page/section) when possible”
    - Upload files, call model, receive comment text
    - Delete files after response

Canvas API mapping (what to call)

You’ll typically use these Canvas REST docs:

- **Submissions API** (get submission details, attachments; also where “grade/comment” operations live) (Instructure Developer Documentation)
- **Rubrics API** (retrieve rubric objects / used locations) (Instructure Developer Documentation)
- **Assignments API** (due dates & assignment metadata) (Canvas)
- **LTI tools + Deep Linking docs** (placement + configuration UX) (Instructure Developer Documentation)

Posting the comment: In Canvas this is done through the Submissions API flow (grade/comment on submission). (Mitt UiB)

FairMark — Practical Implementation Plan (Python)

0. What you are actually building (clear scope)

FairMark = an LTI 1.3 tool + Canvas API consumer + LLM wrapper

It will:

- Trigger after submission (polling or manual run)
- Fetch assignment + rubric + submission
- Apply deterministic policy (late check)
- Send **one structured prompt** + files to LLM
- Receive **one structured evaluation comment**
- Post **comment back to the submission**
- Instructor decides final grade

❌ No auto-grading
❌ No rubric writeback (MVP)
❌ No storing student files

---

1. Tech stack (keep it simple)

Backend (core)

- **Python 3.11**
- **FastAPI** (API + LTI endpoints)
- **uvicorn**
- **SQLAlchemy + PostgreSQL** (only metadata, no files)

Auth & LMS

- **LTI 1.3 (OIDC)**
- Canvas REST API
- OAuth2 (Canvas access token per tenant)

LLM

- OpenAI API (file upload + prompt)

- Strict system prompt

File handling

- Temporary file store (`/tmp`)
- Auto-delete after evaluation

---

2. System modules (actual folders)

fairmark/

```
├── app/
│   ├── main.py              # FastAPI entry
│   ├── lti/
│   │   ├── launch.py        # OIDC + JWT validation
│   │   ├── jwks.py
│   ├── canvas/
│   │   ├── client.py        # Canvas API wrapper
│   │   ├── assignments.py
│   │   ├── submissions.py
│   │   ├── rubrics.py
│   ├── policy/
│   │   ├── parser.py        # admin policy PDF/text
│   │   ├── late_rules.py    # deterministic late logic
│   ├── evaluation/
│   │   ├── packet_builder.py
│   │   ├── prompt_builder.py
│   │   ├── llm_client.py
```

```
│   │   ├── formatter.py
│   ├── db/
│   │   ├── models.py
│   │   ├── session.py
│   └── utils/
│       ├── file_utils.py
│       ├── text_extract.py
```

---

3. Canvas integration (REAL endpoints)

Required Canvas APIs

You **do not** need everything.

Get assignment

GET /api/v1/courses/{course_id}/assignments/{assignment_id}

Includes:

- instructions
- due_at
- points_possible
- rubric (sometimes embedded)

Get rubric (if separate)

GET /api/v1/courses/{course_id}/rubrics

Get submission

GET /api/v1/courses/{course_id}/assignments/{assignment_id}/submissions/{user_id}

Includes:

- submitted_at
- attachments (file URLs)
- submission_id

Download submission file

GET {attachment.url}

Authorization: Bearer CANVAS_TOKEN

Post comment back

POST
/api/v1/courses/{course_id}/assignments/{assignment_id}/submissions/{user_id}/comments

Body:

{

"comment[text_commentary]": "FairMark evaluation comment here"

}

✅ That's it. No grade APIs needed.

---

4. LTI 1.3 — minimal but correct

LTI launch flow

1. Canvas launches FairMark via OIDC
2. FairMark validates:
    - issuer
    - client_id
    - deployment_id

3. Extract:
    - course_id
    - assignment_id
    - user_id
    - role (Instructor / Student)

Store **only IDs**, not content.

---

5. Admin setup (once per university)

Admin UI:

- Upload **University Policy Pack** (PDF or text)
- Define late rules (optional structured form)
- Store Canvas API token

DB table:

tenants (

id,

issuer,

client_id,

canvas_base_url,

canvas_token,

policy_text,

late_rules_json

)

---

6. Deterministic late policy (NO LLM)

def compute_late_tier(due_at, submitted_at, policy):

```
    delta_hours = (submitted_at - due_at).total_seconds() / 3600
    for tier in policy["tiers"]:
        if tier["min"] <= delta_hours <= tier["max"]:
            return tier
    return None
```

Output example:

```
{
  "late": true,
  "hours": 26,
  "penalty_percent": 20
}
```

LLM **never decides penalties**.

---

7. Evaluation packet (actual structure)

```
{
  "submission_meta": {
    "submission_id": "123",
    "submitted_at": "...",
    "due_at": "...",
    "late_tier": "24–48h (-20%)"
  },
  "assignment": {
```

```
    "title": "Week 4 Project",
    "instructions": "...",
    "points": 50
  },
  "rubric": [
    { "name": "Agenda", "points": 5 },
    { "name": "Introduction", "points": 5 }
  ],
  "policy": "University late & grading policy text",
  "week_slides": "optional PDF text",
  "textbook_reference": "Title + citation only"
}
```

---

8. LLM prompt (STRICT, production-ready)

System message

You are FairMark, an AI Teaching Assistant.

Rules:

- Use ONLY the provided sources.

- Do NOT invent missing content.

- If rubric is missing, produce overall evaluation only.

- Missing slides/policy must NOT penalize students.

- Instructor is final authority.

- Output ONE structured evaluation comment.

User message

Includes:

- evaluation packet (JSON)
- uploaded submission file
- uploaded slides (if any)

---

9. Output format (what gets posted)

Overall Evaluation:

This submission demonstrates strong coverage of the core requirements with minor clarity and formatting issues.

Agenda — 5/5

Comment: Very clear and complete agenda slide.

Introduction — 4.5/5

Comment: The introduction is clear, but it should more explicitly explain why a risk management plan benefits TechWorx Resale.

Threats and Vulnerabilities — 5/5

Comment: Two relevant threats and vulnerabilities are clearly identified.

...

Possible Final Grade (before instructor review): 47 / 50

Note:

This is a pre-evaluation generated by FairMark. Your instructor will review and finalize the grade.

---

10. Trigger strategy (MVP)

Easiest & reliable

- **Instructor clicks “Run FairMark”**
- Or **cron polling every 5 minutes**:

GET submissions where evaluated = false

No webhook complexity initially.

---

11. API calls to LLM (example)

```
client.responses.create(
  model="gpt-4.1",
  input=[{
    "role": "user",
    "content": [
      {"type": "input_text", "text": prompt_text},
      {"type": "input_file", "file_id": submission_file_id}
    ]
  }]
```

)

---

12. Development timeline (realistic)

Week	Deliverable
1	LTI 1.3 launch + Canvas auth
2	Fetch assignment + submission
3	Policy + late logic
4	LLM evaluation + comment
5	Post comment + cleanup
6	Instructor UI + polish

Demo assessment response (your requested structure)

Overall Evaluation (short):
Good work overall—your slides are clear and well-structured, and your compliance section is strong. A few items need refinement for maximum credit: strengthen the **benefit justification** in the introduction, include **mid-level managers** in roles/responsibilities, and rewrite risk statements as readable **if–then statements** (not diagrams). Your schedule is close but needs recurring tasks grouped in a single row format.

Rubric Breakdown

- **Agenda — 5/5**
  Comment: Very good. Agenda is clear and matches the flow of the presentation.
- **Introduction — 4.5/5**
  Comment: Good start, but explicitly explain **why** a risk management plan is beneficial for **TechWorx ReSale** (e.g., reduces financial loss, improves compliance readiness, supports business continuity).
- **Threats and Vulnerabilities — 5/5**
  Comment: Strong choices and aligned with the scenario. Some supporting images are too small to read—consider enlarging or simplifying them.
- **Compliance Laws — 5/5**
  Comment: Very good—relevant laws/regulations and well summarized.
- **Key Roles and Responsibilities — 4.5/5**
  Comment: Add mid-level managers from **HR, Operations, and Logistics** to make the organizational view complete.
- **Risk Statements — 3.5/5**
  Comment: The if–then risk statements should be written as **clear text statements**, not diagrams. Also increase readability—avoid forcing the grader to zoom.
- **Develop Schedule — 4.5/5**
  Comment: Good schedule overall. Recurring activities (e.g., periodic reviews/IPRs) should appear on **one row** with repeated markers to show recurrence.
- **Grammar / Clear Slides / Supporting Images — 5/5**
  Comment: Clear writing and generally professional slides.
- **Summary Slide — 5/5**
  Comment: Strong summary that closes the presentation well.
- **References and Inline Citation — 5/5**
  Comment: Citations and references are complete and consistent.

Possible Final Grade: 47/50

Dataset / Study plan (what you will collect & report)

Field	What to record	Why it matters
Courses	#courses, discipline, level (UG/Grad), class size	generalizability
Assignments	#assignments, type (essay/report/slides/lab), points	controls difficulty
Rubrics	#criteria, points per criterion, rubric availability rate	core input quality
Submissions	#submissions, attempts, on-time vs late	policy + realism
File formats	PDF/DOCX/PPTX distribution	robustness
Artifacts available	policy pack available (admin), week slides available rate	ablation / reliability
Ground truth	instructor final rubric scores + final grade	agreement metrics
Time	instructor grading time (baseline vs FairMark-assisted)	productivity impact
Outcomes	instructor edit rate on FairMark comment	usefulness
Privacy	anonymization method, retention windows	ethics & compliance

---

Primary evaluation metrics (journal style)

Metric	Definition	Report
Criterion MAE	mean $\lvert AI\_points - instructor\_points \rvert$ per criterion	mean ± std
Total MAE	mean $\lvert AI\_total - instructor\_total \rvert$	mean ± std
Over-score rate	% where AI_total > instructor_total by > δ	%
Under-score rate	% where AI_total < instructor_total by > δ	%
Rank correlation	Spearman ρ between AI_total and instructor_total	ρ
Rubric coverage	% criteria with a verdict + comment	%

Evidence compliance	% criteria with explicit “evidence pointer” OR “not enough evidence”	%
Latency	submission → comment posted time	median, p95
Instructor time saved	baseline time − assisted time	minutes/submission
Instructor acceptance	% comments used with minimal edits	%
