# Embedding In-Browser Linux Labs as Smart Content for Intelligent Textbooks

**Andrzej Wójtowicz**

Adam Mickiewicz University, Poznań, Poland

andre@amu.edu.pl

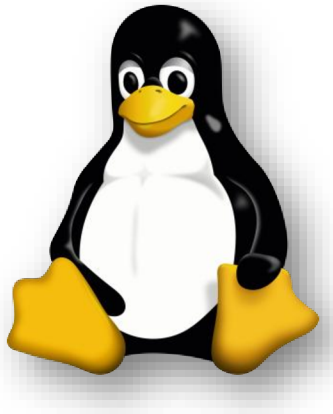
# Outline

1. Background
2. Demo
3. Design
4. AI extension
5. Limits

# Background (1)

”Intro to Linux” course:

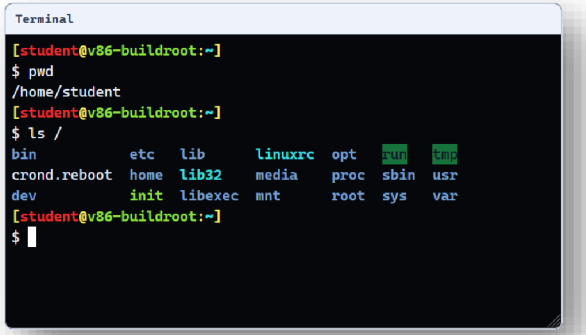
- Mandatory course for computer science undergraduates, 1st semester.
- Learning mostly Bash.
- Textbook as HTML static content.
- Working with Linux in class and… at home?



# Background (2)

Looking for a tool where learners can:

- inspect files,
- make mistakes safely,
- recover from those mistakes,
- get feedback on the real system state.




Usual tradeoff:

**Local VM**

- Good control
- No hosting cost
- Hard setup for beginners

**Hosted lab**

- Central control
- Provider cost and operations
- Easy start for learners

… or maybe something embedded and self-contained in a textbook?

# Background (3)

Core idea:

- Treat a Linux lab as smart textbook content.
- Not a link to an external service.
- Not a setup instruction that sends students away.
- A bootable, inspectable learning object embedded next to the lesson.

Design constraints:

- The server serves static files.
- The learner’s browser does the execution.

# Demo

1. Open the textbook lesson excerpt.
2. Load first activity.
3. Build a shell pipeline.
4. Run the local checker.

# Chapter 4. Pipes and Log Processing

A central idea in Unix-like systems such as Linux is to combine several small, simple commands to produce a more complex result, rather than rely on a single program with many options and modes of operation. Pipes make this possible by sending the standard output of one command to the standard input of the next. Commands in a pipeline are connected with the vertical bar symbol `|`; for example: `command1 | command2 | command3`.

The commands in a pipeline can process data as it becomes available, so later commands do not always need to wait for the entire preceding command to finish. This makes pipelines especially useful for concise and efficient data processing.

In log-processing tasks, this lets you build small, inspectable transformations instead of writing a full program. A typical workflow is to filter relevant lines with `grep`, select the field you need with `cut`, order the values with `sort`, and count repeated values with `uniq -c`.

## Practice Activity 1: WARN by service

Count how many `WARN` log entries belong to each service. Save a sorted frequency report in `warn-by-service.txt`.

Load VM | WORKSPACE: /home/student/pipes-lab-1 | INPUT: app.log | OUTPUT: warn-by-service.txt

## Practice Activity 2: Failed login users

Extract the unique usernames from failed authentication lines. Save one sorted username per line in `failed-users.txt`.

Load VM | WORKSPACE: /home/student/pipes-lab-2 | INPUT: auth.log | OUTPUT: failed-users.txt

## Practice Activity 3: Top endpoints

Count requested endpoints that returned a `5xx` status code. Save the frequency table in `top-5xx-endpoints.txt`, highest count first.

Load VM | WORKSPACE: /home/student/pipes-lab-3 | INPUT: access.log | OUTPUT: top-5xx-endpoints.txt

## Practice Activity 1: WARN by service

Count how many `WARN` log entries belong to each service. Save a sorted frequency report in `warn-by-service.txt`.

Reset VM | WORKSPACE: /home/student/pipes-lab-1 | INPUT: app.log | OUTPUT: warn-by-service.txt

Terminal — virtio-console0

```
Welcome to Buildroot
Pipes lab 1: WARN by service

Workspace: /home/student/pipes-lab-1
Task: read README.txt, process app.log, write warn-by-service.txt.
Use the textbook Check button when ready.
[student@v86-buildroot:~/pipes-lab-1]
$ ls
README.txt  app.log
[student@v86-buildroot:~/pipes-lab-1]
$ head -3 app.log
2026-06-01T09:00:02Z|INFO|api|request accepted
2026-06-01T09:01:17Z|WARN|auth|password retry threshold reached
2026-06-01T09:02:44Z|WARN|api|slow upstream response
[student@v86-buildroot:~/pipes-lab-1]
$
```

Check answer | Show hint | Show solution

STATUS ready

## Practice Activity 1: WARN by service

Count how many `WARN` log entries belong to each service. Save a sorted frequency report in `warn-by-service.txt`.

Reset VM | WORKSPACE: /home/student/pipes-lab-1 | INPUT: app.log | OUTPUT: warn-by-service.txt

**Terminal** virtio-console0

```
Pipes lab 1: WARN by service

Workspace: /home/student/pipes-lab-1
Task: read README.txt, process app.log, write warn-by-service.txt.
Use the textbook Check button when ready.
[student@v86-buildroot:~/pipes-lab-1]
$ ls
README.txt  app.log
[student@v86-buildroot:~/pipes-lab-1]
$ head -3 app.log
2026-06-01T09:00:02Z|INFO|api|request accepted
2026-06-01T09:01:17Z|WARN|auth|password retry threshold reached
2026-06-01T09:02:44Z|WARN|api|slow upstream response
[student@v86-buildroot:~/pipes-lab-1]
$ grep '|WARN|' app.log | cut -d'|' -f3 | sort | uniq -c > warn-by-service.txt
[student@v86-buildroot:~/pipes-lab-1]
$
```

Check answer | Show hint | Show solution

STATUS passed

**Feedback**

OK: warn-by-service.txt has the expected WARN counts.

```
$ grep '|WARN|' app.log | cut -d'|' -f2 | sort | uniq -c > warn-by-service.txt
[student@v86-buildroot:~/pipes-lab-1]
$
```

Check answer | Show hint | Show solution

STATUS needs work

**Feedback**

```
ERROR: warn-by-service.txt does not match the expected WARN counts.
Expected lines:
2 api
3 auth
2 billing
1 worker
Diff:
--- /tmp/tmp.sBocIOVveA 2026-06-21 20:39:09.157000000 +0000
+++ /tmp/tmp.6IdvSTPOT2 2026-06-21 20:39:09.200000000 +0000
@@ -1,4 +1 @@
-2 api
-3 auth
-2 billing
-1 worker
+8 WARN
```

## Debug

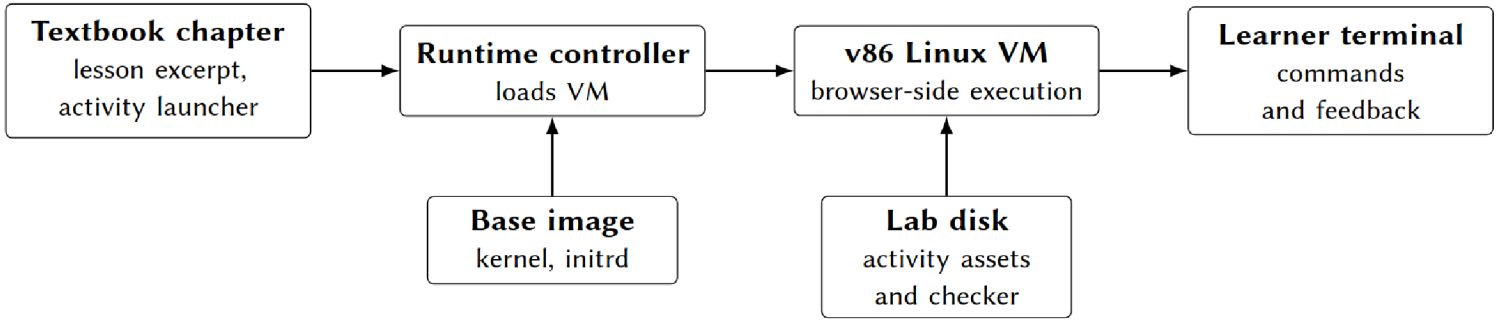
Hidden `serial1` control channel used by the checkers.

**Terminal** serial1

```
__CHECK_lab01_1782074349216_5icyz14wpps__:START
[root@v86-buildroot:~]
# /bin/sh /usr/local/bin/check-current-lab
ERROR: warn-by-service.txt does not match the expected WARN counts.
Expected lines:
2 api
3 auth
2 billing
1 worker
Diff:
--- /tmp/tmp.sBocIOVveA 2026-06-21 20:39:09.157000000 +0000
+++ /tmp/tmp.6IdvSTPOT2 2026-06-21 20:39:09.200000000 +0000
@@ -1,4 +1 @@
-2 api
-3 auth
-2 billing
```

# Design (1)

Architecture:




Deployment model:

- HTML + CSS + JavaScript
- VM:
  - Buildroot-based Linux *base image* (kernel+initrd) and *lab disks*
  - v86 browser-based x86 emulator (WebAssembly) with Xterm.js

# Design (2)

Base image:

- Reusable Linux system
- Kernel, initrd, tools, users
- Shared across a textbook module

Lab disk:

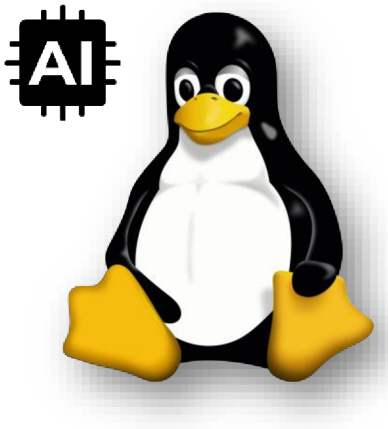
- Small task-specific image for a given practice activity
- Datasets and scripts
- Checker for one exercise

# AI extension

Optional in the current demonstrator.

Does not generate full Linux image:

- Generate a compact activity specification
- Use module base Linux image.



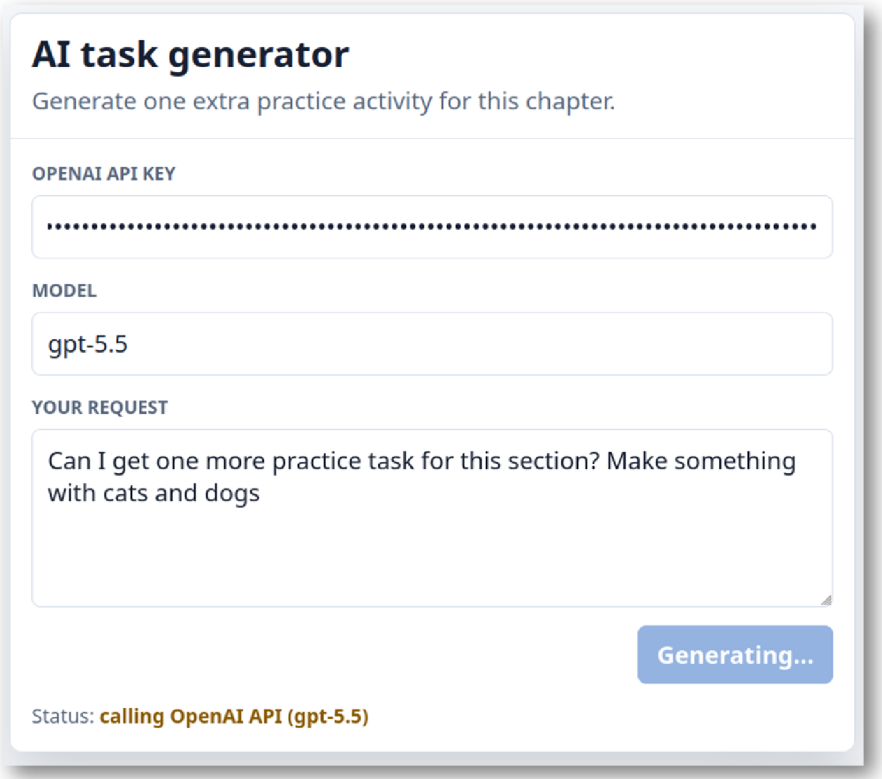

Two options:

**In-browser SLM**

- No hosting cost
- Limited by device and model quality
- Not reliable enough with Qwen2.5-Coder-0.5B-Instruct

**Hosted/institutional LLM**

- It may cost a lot…
- Better quality today


AI task generator
Generate one extra practice activity for this chapter.
OPENAI API KEY
MODEL
gpt-5.5
YOUR REQUEST
Can I get one more practice task for this section? Make something with cats and dogs
Generating...
Status: calling OpenAI API (gpt-5.5)

## Practice Activity AI: Sent pet reminders

In `pet-notify.log`, count how many cat and dog entries contain `event=reminder status=sent` by animal type, and save the counts in `sent-reminders-by-pet.txt`.

Reset VM | WORKSPACE: /home/student | INPUT: pet-notify.log | OUTPUT: sent-reminders-by-pet.txt

Terminal — virtio-console0

```
[student@v86-buildroot:~]
$ ls
pet-notify.log
[student@v86-buildroot:~]
$ head -3 pet-notify.log
2026-05-02T09:00:01Z clinic=west pet=cat event=reminder status=sent owner=ava
2026-05-02T09:01:18Z clinic=west pet=dog event=reminder status=bounced owner=ben
2026-05-02T09:03:44Z clinic=east pet=cat event=reminder status=sent owner=cleo
[student@v86-buildroot:~]
$ grep 'event=reminder status=sent' pet-notify.log | cut -d' ' -f3 | cut -d= -f2 | sort | uniq -c > sent-reminders-by-pe
t.txt
[student@v86-buildroot:~]
$
```

Check answer | Show hint | Show solution — STATUS: passed

**Feedback**

OK

# Limits (1)

Feasibility checks:

- Buildroot base image used in testing: 38 MB.
  - Can be optimized to 13 MB.
- Typical text-based activity lab disk: 100 KB.

Boot time depends on image size, browser, and learner device.

# Limits (2)

- Not a classroom evaluation.
- Not secure summative assessment.
- Not robust unsupervised LLM/SLM-generated shell scripting.
- Not a full LMS integration.

# Thank you!