DATA WYDANIA: 06.2025

ŚCIEŻKA KARIERY PROGRAMISTA **GEN-AI**

VS CODE, POETRY, GIT, JUPYTER, DOCKER | PYTHON, NUMPY, PYTORCH, TRANSFORMERS | HUGGING FACE, DATASETS, PEFT, LORA, QLORA | STABLE DIFFUSION, CONTROLNET, WHISPER, AUDIOCRAFT, STARCODER | PROMPT ENGINEERING, RAG, RLHF, EVALUATE | FASTAPI, STREAMLIT, DOCKER, GITHUB ACTIONS, DVC | CI/CD, WANDB, MLFLOW, BITSANDBYTES, VLLM | MONITORING, SAFETY, WATERMARKING, GUARDRAILS | KOMPLETNE PORTFOLIO

Spis Treści

Wstęp

Ludziom często się wydaje, że AI tworzy niesamowite obrazy, kreuje coś z niczego. Tymczasem prawda jest taka, że “sztuka AI” to iloczyn gigantycznej ilości parametrów powstały w oparciu o istniejące dane — obrazowe i tekstowe. Za każdy, obrazkiem stworzonym przez Dall-E czy Midjourney stoi sztab specjalistów od danych, który przygotował obrazy, dodał do nich tagi czy etykiety, a także jeszcze większy sztab specjalistów od NLP, który odpowiednio zwektoryzował prompt tak, by obraz generowany na jego podstawie odpowiadał temu, co człowiek myśli, że powinien generować.

Śmiem twierdzić, że Gen-AI to najbardziej różnorodna specjalizacja AI, ponieważ warstwa wejściowa (zazwyczaj językowa lub dane złożone - prompt + dodatkowe dane) oraz warstwa wyjściowa (obraz, dźwięk lub tekst) mają często różną formę. Przez to programista specjalizujący się w Gen-AI musi być specjalistą w obu tych dziedzinach.

Gen-AI jest trochę jak połączenie CV i NLP. Ale nie mów tego głośno, bo możesz przypadkiem obrazić wielu specjalistów od CV lub NLP. Ale prawda jest taka, że jeśli zajrzysz do ścieżek kariery dla obu tych specjalizacji, znajdziesz wiele elementów, które poznasz również w ścieżce kariery dla Gen-AI.

Programista Gen-AI musi więc umieć pisać prompty, stroić adaptery LoRA, kontrolować generację obrazów i serwować to wszystko w jednej usłudze. To praktyczny punkt przecięcia NLP, Computer Vision i inżynierii danych multimodalnych, wzbogacony o prompt-engineering i lekki MLOps.

Ta ścieżka kariery daje większość fundamentów z CV i NLP, ale uczy też łączenia ich w gotowe produkty. Nasz e-book przeprowadzi Cię przez wszystkie etapy — od instalacji środowiska i podstaw matematyki, przez trenowanie LLM-ów i modeli Diffusion, aż po wdrożenie, monitoring i testów etyczne.

Każdy miesiąc ma swój temat przewodni, zadania i kamień milowy. Przez cały rok budujesz repozytorium na GitHubie, które będzie jednocześnie Twoim portfolio. Na koniec roku znajdziesz tam kompletne projekty z modelami, metrykami i kompletnym pipeline’em.

Aby to osiągnąć, musisz poświęcić średnio 8–10 godzin tygodniowo, trochę samozaparcia i wytrwałości, by bez stresu “skakać” pomiędzy różnymi elementami układanki (to naprawdę taki “fullstack AI”).

O resztę zadbamy my — na portalu porozmawiAImy.pl znajdziesz podstronę dla każdego miesiąca, notatki, materiały do druku i pomoce, dzięki którym nauka będzie znacznie prostsza. A kiedy spadnie Ci motywacja — zajrzyj na bloga Projekt10K i ponarzekaj razem z nami na bias w ChacieGPT i halucynacje MidJourney'a.

To co? Gotowi na rok z Gen-AI? Zaczynamy!

Jak pracować z eBookiem?

Zakładamy, że w każdym miesiącu jesteś w stanie poświęcić 8-10h/tygodniowo (choć większość zadań powinna zająć mniej czasu). Jeśli zdarzy Ci się dodatkowy czas wolny, który chcesz poświęcić na naukę lub skończysz zadania przed czasem, możesz wziąć się za zadania dodatkowe – na każdy miesiąc przygotowaliśmy zadania dodatkowe na 1 wolny weekend i na 2 wolne weekendy (wymagają odpowiednio po ok. 8h i 16h).

Swoje notatki (tzw. notki techniczne) przechowuj w repo, które stworzymy w pierwszy miesiącu. Pisanie swoich spostrzeżeń to dobry sposób na zapamiętywanie informacji (podobno powoduje przenoszenie informacji z pamięci krótkotrwałej do pamięci długotrwałej). A wrzucanie commitów na GitHub pomoże Ci poćwiczyć sprawne korzystanie z Gita.

Na koniec każdego miesiąca na Twoim repo powinny się znaleźć następujące elementy:

- journal/miesiąc-X.md - plik z notatkami wysłany na repo
- notebooks/ - do tego folderu wrzucamy wszystkie pliki .ipynb (jeśli projekt jest bardziej złożony, załóż dla niego oddzielne repo, a notatki i spostrzeżenia wyciągnięte w trakcie pracy wrzuć do folderu journal.
- README.md - to główny plik z dokumentacją projektu. Aktualizuj go co miesiąc (Warto, bo to będzie podstawa Twojego portfolio).

Miesiąc I. — Setup & Flow dla Programisty Gen-AI

W tym miesiącu stworzymy warsztat pracy: zainstalujemy Pythona 3.12, skonfigurujemy środowisko, wrzucimy repo na GitHuba i przygotujemy szkielet projektu na resztę roku. Porównaj to z szykowaniem domu przed przeprowadzką — w końcu jak sobie pościelesz, tak się wyśpisz. Od teraz każda linijka kodu będzie mieć swoje miejsce, wersję, log i dokumentację.

1. Instalacja i konfiguracja środowiska — 4 h

- **Co robisz:** instalujesz Python 3.12, Miniconda/Poetry, VS Code (ext: Python, Jupyter), na GPU: sterownik NVIDIA + CUDA 12, nvidia-smi test, zakładasz repo sciezka-kariery-genai z plikami .gitignore, README.md
- **Materiały:** oficjalne install-guides (Python, CUDA) + wideo „*VS Code for Data Sci*".
- **Cel:** python -m pip list zwraca torch, transformers; git push origin main działa.

2. Git od zera — 6 h

- **Co robisz:** kurs *„Git & GitHub Crash Course*", robisz 5 commitów, jeden Pull Request do własnego repo (branch feature/readme-logo).
- **Materiały:** kanal „KodeGit" (YouTube, 45 min).
- **Cel:** zielony PR merge + opis commitów w journal/.

3. Python „idioms quick-win" — 6 h

- **Co robisz:** mini-kurs: list-/dict-comprehensions, type-hints (mypy --strict), dataclasses, argparse.
- **Materiały:** „*Effective Python*" Rozdz. 1–4.
- **Cel:** skrypt tokenize.py przyjmuje tekst i zwraca tokeny transformers (CLI).

4. Poetry + pre-commit — 3 h

- **Co robisz:** poetry init; dodajesz black, ruff, pre-commit, konfigurujesz hooki.
- **Materiały**: docs.poetry, blog „Pre-commit for ML".
- **Cel:** git commit formatuje kod bez błędów lint.

5. Notebook „Hello NumPy → Torch” — 4 h

- **Co robisz:** konwertujesz macierz NumPy → Tensor, liczysz prosty iloczyn, włączasz GPU (tensor.to("cuda")).
- **Cel:** notatnik notebooks/hello_torch.ipynb z wykresem czasu CPU vs GPU.

6. CLI & Bash podstawy — 4 h

- **Co robisz:** tutorial „Bash-Fu 101”, piszesz skrypt setup_env.sh (instalacja venv + libs).
- **Cel:** ./setup_env.sh od zera buduje środowisko na świeżej maszynie.

7. Podsumowanie & retrospektywa — 2 h

- **Co robisz:** plik journal/miesiac-1.md z listą: co działa, co sprawiło trudność, TODO na M-II.
- **Cel:** commit „M-I: środowisko gotowe”.

Jeśli masz ekstra wolny weekend (8h)

Docker „Hello LLM” – 4 h

- **Co robisz:** Budujesz pierwszy obraz Dockera o nazwie *genai-infer:dev*, który zawiera środowisko z Pythonem 3.12, biblioteką transformers i serwerem uvicorn. Tworzysz prosty serwis FastAPI z jednym endpointem GET /health, który zwraca {"status": "ok"}.
- **Materiały:** FastAPI – tutorial "First Steps", Docker – Python + FastAPI quickstart;
- **Cel:** kontener działa lokalnie, uruchamia się w < 5 s, a testowy request zwraca poprawną odpowiedź.

GitHub Actions CI – 4 h

- **Co robisz:** ruff check . (lint), pytest (testy jednostkowe), docker build obrazu, zapisuje artefakt z obrazem jako cache lub release asset
- **Materiały:** GitHub Actions: starter templates + „Python CI with Docker”, Wzorzec z e-booka FinTech AI (moduł: CI/CD minimalny)
- **Cel:** zielony ✔ workflow na GitHubie przy pushu, plik workflow + badge w README.md.

Jeśli masz 2 ekstra wolne weekendy (16h)

Automatyczne formatowanie kodu – 6 h

- **Co robisz:** Konfigurujesz pre-commit z hookami: black --check do sprawdzania formatowania, ruff --fix do automatycznego lintowania, opcjonalnie end-of-file-fixer, trailing-whitespace. Włączasz uruchamianie hooków lokalnie oraz w CI (ci.yml). Dodajesz badge „lint passing" do README.md.
- **Materiały:** pre-commit.com (config + lista hooków), dokumentacja black i ruff, e-book NLP: sekcja „pre-commit jako firewall jakości kodu"
- **Cel:** kod repo zawsze przechodzi pre-commit lokalnie i na CI, workflow odpala lint + fix automatycznie.

Głębszy kurs Bash – 6 h

- **Co robisz:** Tworzysz dwa przydatne skrypty: *backup.sh* – kopiuje folder notebooks/ do backups/YYYY-MM-DD/; *gpu_watch.sh* – wypisuje nvidia-smi co 5 s z timestampem. Edytujesz .bashrc lub .zshrc i dodajesz aliasy do najczęstszych komend (gac, nb, serve). Wykorzystujesz chmod +x, trap, tee i while true.
- **Materiały:** Bash Handbook (explainshell.com + tldr.sh), YouTube: „Practical Bash Scripts for ML Devs", e-book CV: „Bash-samouczek dla AI-deva"
- **Cel:** folder scripts/ zawiera działające backup.sh, gpu_watch.sh; aliasy zapisane w .bashrc; plik scripts/README.md z opisem.

Notka techniczna – 4 h

- **Co robisz:** Przygotuj wewnętrzny plik *notes/poetry_vs_requirements.md*, w którym porównasz Poetry i klasyczny requirements.txt. Opisz, jak działają zależności, lockfile i workflow z pre-commit. Dołącz przykładowe komendy i najczęstsze błędy z rozwiązaniami.
- **Cel**: plik gotowy do późniejszego użytku jako ściąga lub część dokumentacji projektu.

Kamień milowy I. miesiąca

Pierwszy miesiąc za Tobą — masz już nie tylko działające środowisko, ale też repozytorium gotowe do pracy zespołowej, z automatycznym formatowaniem

kodu i pierwszymi własnymi skryptami. To punkt wyjścia, który za chwilę rozbudujesz o modele i aplikacje.

Repozytorium *genai-playground* zawiera:

- działające środowisko (Python 3.12 + CUDA + Torch),
- workflow CI: lint (ruff), testy (pytest), build Docker,
- pliki: README.md, journal/miesiac-1.md, setup_env.sh,
- folder scripts/ z backup.sh, gpu_watch.sh + scripts/README.md,
- plik konfiguracyjny .pre-commit-config.yaml + aktywne hooki,
- (bonus) kontener genai-infer:dev dostępny w Docker Hub.

Gotowe? Brawo. To nasz fundament, na którym zbudujemy cały stack LLM-ów, RAG-ów i innych multimodalnych cudów.

Miesiąc II. — Python Core + Matematyka & klasyczne ML pod Gen-AI

W tym miesiącu wracamy do podstaw. Albo raczej do fundamentów. Matematycznych fundamentów, które są przydatne w pracy z modelami: algebry liniowej, rachunku różniczkowym i podstaw statystyki. Poznamy też klasyczne techniki uczenia maszynowego i zbudujemy nasz pierwszy pipeline scikit-learn do analizy tekstu. Pod koniec miesiąca będziesz mieć NumPy w małym palcu, a gradienty nie będą mieć przed Tobą tajemnic.

1. Szybki kurs Python Advanced — 6 h

- **Co robisz:** przerabiasz rozdziały 5-8 *„Effective Python”* (context manager, itertools, funtools).
- **Materiały:** „Effective Python”, seria wideo *Real Python* (4 × 10 min).
- **Cel:** skrypt matrix_utils.py wykorzystuje zip_longest i with open() jako kontekst.

2. 50 zadań code-kata (Exercism / LeetCode) — 6 h

- **Co robisz:** rozwiązujesz 50 prostych tasków, commit po każdym bloku 10.
- **Cel:** badge „50 Solved” na koncie LeetCode; link w README.

3. Algebra liniowa w NumPy — 6 h

- **Co robisz:** implementujesz svd_demo.ipynb (rozklad SVD + wizualizacja).
- **Materiały**: 3Blue1Brown Essence of Linear Algebra #7-#12.
- **Cel:** heat-map singular values + opis w notatniku.

4. Rachunek różniczkowy & gradient descent — 6 h

- **Co robisz:** notebook gradient_linreg.ipynb — regresja liniowa z własnym GD.
- **Materiały:** StatQuest „Gradient Descent”, rozdz. 4 Goodfellow.
- **Cel:** loss < 0.01 na zbiorze Boston Housing.

5. Statystyka & p-value quick-win — 4 h

- **Co robisz:** skrypt t_test.py porównuje dwie próbki; używa scipy.stats.
- **Cel:** README tłumaczy kiedy użyć testu t i rysuje histogram.

6. Pipeline scikit-learn (baseline NLP) — 4 h

- **Co robisz:** sklearn_pipeline.ipynb — TF-IDF + LogReg na POL-tweets.
- **Materiały:** oficjalny tutorial "From text to feature".
- **Cel:** accuracy ≥ 0,80; zapis modelu baseline.joblib.

7. Retrospektywa & journal — 2 h

- **Co robisz:** journal/miesiac-2.md z TOP3 sukcesy, TOP3 problemy, plan M-III.
- **Cel:** commit „M-II: NumPy & Math ready".

Jeśli masz ekstra wolny weekend (8h)

Micrograd-style autograd — 6 h

- **Co robisz:** Na podstawie mini-projektu Andreja Karpathy'ego (micrograd) implementujesz własny silnik autograd: klasa Value, operator przeciążający +, *, tanh, backward() i wizualizacja grafu obliczeń. Tworzysz test: sieć XOR (2 → 2 → 1) trenowana przez 1000 epok, loss MSE < 0.05. Kod trzymasz w libs/micrograd_clone.py, testy w tests/test_micrograd.py.
- **Materiały:** Repozytorium Karpathy'ego, YouTube: „Let's build micrograd from scratch" (15 min), Blog: „Autograd without magic" (Eli Bendersky)
- **Cel:** w pełni działająca mikroimplementacja autograd, działający test XOR, log lossu w journal/.

Notka techniczna — 2 h

- **Co robisz:** Tworzysz plik notes/numpy_broadcasting.md, w którym wyjaśniasz zasadę działania broadcastingu w NumPy na 3–4 przykładach: sumowanie macierzy i wektora, mnożenie przez skalar, dopasowanie wymiarów (np. A[:, np.newaxis] * B) Wyjaśniasz, czym jest zasada „trailing dimensions" i jak broadcast wpływa na wydajność. Dodajesz checklistę „Czy to zadziała?" z 5 przypadkami „true/false".

- **Materiały:** Dokumentacja NumPy: Broadcasting, Blog: „Understanding NumPy Broadcasting by Example"
- **Cel:** gotowy plik numpy_broadcasting.md jako ściąga do dalszych projektów.

Jeśli masz 2 ekstra wolne weekendy (16h)

Pakiet matrix_utils + Poetry publish — 6h

- **Co robisz:** Tworzysz własny moduł matrix_utils/ z funkcjami przydatnymi do operacji macierzowych w ML: normalize_rows, softmax_matrix, cosine_similarity_batch. Dodajesz testy pytest w folderze tests/, piszesz pyproject.toml, konfigurujesz build i publish do TestPyPI. Generujesz plik matrix_utils-0.1.0.tar.gz, testujesz lokalną instalację w osobnym env.
- **Materiały:** Oficjalna dokumentacja Poetry: Publishing Packages, Blog „How to create and test your own Python package", Przykład z e-booka CV: moduł image_ops (style: kod + testy + release)
- **Cel:** gotowy pakiet matrix_utils opublikowany na TestPyPI, test instalacji działa (pip install --index-url https://test.pypi.org/simple matrix-utils), README z opisem funkcji.

Projekt „TinyGPT-2-scratch" — 10 h

- **Co robisz:** implementujesz uproszczony model GPT-2 z jedną warstwą: Embedding, MultiHeadAttention, FeedForward, LayerNorm, CausalMask. Trenujesz model na korpusie Tiny Shakespeare (tiny_shakespeare.txt, ~1MB), batch size = 16, context = 64, learning rate = 3e-4, tokenizer: character-level. Logujesz loss i generujesz przykładowe outputy.
- **Materiały:** Blog: „GPT from scratch" by Jay Alammar, Repo: nanoGPT (Karpathy) jako inspiracja, Artykuł: „The Illustrated GPT-2"
- **Cel:** loss < 4.0 na TinyShakespeare, plik tinygpt_model.pt zapisany, wygenerowany przykładowy tekst w samples/.
  Dodatkowo: opis architektury i wyniki w notes/tinygpt_summary.md.

Kamień milowy II. miesiąca

Drugi miesiąc to moment, w którym matematyka przestaje być teorią, a zaczyna działać w kodzie. Zbudowaliśmy pierwsze pipeline'y ML, liczymy gradienty i

nareszcie rozumiemy, co robi np.dot – zamiast się domyślać. Na Twoim repo pojawiły się pliki z kodem, ale też testy, notatniki i pierwsze własne narzędzia.

Repozytorium *genai-playground* zawiera:

- bibliotekę matrix_utils/ z testami PyTest i pyproject.toml,
- trzy notatniki: svd_demo.ipynb, gradient_linreg.ipynb, sklearn_pipeline.ipynb,
- badge „50 Solved" (np. z LeetCode/Exercism) + README.md z wynikami testu t,
- wpis journal/miesiac-2.md z retrospektywą miesiąca.

Ten kamień milowy potwierdza, że masz działające fundamenty matematyczno-programistyczne i pierwszy kod gotowy do ponownego użycia — coś, co za chwilę bardzo się przyda w PyTorch i transformerach.

Miesiąc III — PyTorch & Transformers Fundamentals

W tym miesiącu wracamy do podstaw. Albo raczej do fundamentów. Matematycznych fundamentów, które są przydatne w pracy z modelami: algebry liniowej, rachunku różniczkowym i podstaw statystyki. Poznamy też klasyczne techniki uczenia maszynowego i zbudujemy nasz pierwszy pipeline scikit-learn do analizy tekstu. Pod koniec miesiąca będziesz mieć NumPy w małym palcu, a gradienty nie będą mieć przed Tobą tajemnic.

1. Instalacja PyTorch 2.2 GPU — 3 h

- **Co robisz:** pip3 install torch==2.2.* torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121; test torch.cuda.is_available().
- **Materiały:** oficjalna strona *Start Locally*.
- **Cel:** notebook cuda_check.ipynb drukuje nazwę karty i wersję CUDA.

2. Autograd & własny training-loop — 6 h

- **Co robisz:** piszesz train_mnist.py: ręcznie definiowany Linear -> ReLU -> Linear; obliczasz loss F.cross_entropy, pętla GD/Adam, zapis accuracy.
- **Cel:** accuracy ≥ 0,95 na MNIST CPU lub GPU, log w journal/.

3. Hugging Face Transformers quick-start — 6 h

- **Co robisz:** pipeline("sentiment-analysis") na 5 zdaniach PL/EN, skrypt hf_infer.py --model distilbert-base-uncased.
- **Cel:** json-out z probabilities zapisany do outputs/.

4. Fine-tuning BERT (Pol-Emo2.0) — 7 h

- **Co robisz:** używasz Trainer + datasets, 2 epo, batch 16, learning-rate 2e-5.
- **Cel:** F1-macro ≥ 0,85; zapis model/ + eval.json.

5. PEFT intro (LoRA / QLoRA) — 4 h

- **Co robisz:** przerabiasz tutorial Parameter-Efficient Fine-Tuning, adaptujesz model z zadania 4 na LoRA rank 8.
- **Cel:** plik pytorch_model.bin ma < 120 MB.

6. Tracking + eksperymenty — 4 h

- **Co robisz:** integrujesz Weights & Biases (wandb.init(project="genai-path")), logujesz: loss, lr, GPU-mem.
- **Cel:** publiczny run link + zrzut ekranu w README.

7. Retrospektywa & journal — 2 h

- **Co robisz:** journal/miesiac-3.md — sukcesy, blokery, plan na kwiecień.
- **Cel:** commit "M-III: torch & transformers ready".

Jeśli masz ekstra wolny weekend (8h)

Rotary PE + benchmark — 6 h

- **Co robisz:** Implementujesz Rotary Positional Embeddings (RoPE) i podmieniasz klasyczne nn.Embedding na Rotary w swoim MLP z zadania 2 (train_mnist.py). Mierzysz wpływ na: czas treningu (czas epoki), końcowy loss, jakość predykcji (np. accuracy). Wyniki zapisujesz w tabeli rotary_vs_baseline.md.
- **Materiały:** Blog: RoFormer & Rotary Positional Embedding explained; Repozytorium: rotary-embedding-torch (https://github.com/lucidrains/rotary-embedding-torch); Paper: RoFormer: Enhanced Transformer with Rotary Position Embedding
- **Cel:** działająca implementacja RoPE, porównanie wyników, tabelka rotary_vs_baseline.md.

Mini-post — 2 h

- **Co robisz:** Piszesz wpis journal/rotary_pe.md z krótkim wytłumaczeniem: czym jest Rotary PE, jak różni się od klasycznego sin/cos, kiedy warto go używać. Dodajesz przykładowy wykres lub snippet kodu.
- **Cel:** gotowy wpis rotary_pe.md jako ściąga i podsumowanie eksperymentu.

Jeśli masz 2 ekstra wolne weekendy (16h)

Fine-tune Llama-3-8B-Instruct (LoRA) — 12 h

- **Co robisz:** Przygotowujesz zbiór qa_data.jsonl (~10 000 przykładów: {"question": "...", "answer": "..."}), Konfigurujesz fine-tuning: peft + transformers, LoRA rank=16, batch 2, gradient accumulation 16, 3 epoki, model: meta-llama/Meta-Llama-3-8B-Instruct, użycie 1 GPU 24 GB lub T4 z gradient checkpointingiem. Po zakończeniu: zapis adaptera .bin, upload do Hugging Face Hub (private repo), napisanie README.md i model_card.md.
- **Materiały:** Hugging Face PEFT + Transformers docs, Blog: Fine-tune LLaMA 3 on Custom Data with LoRA, Dataset generator (opcjonalnie): self-instruct style generator
- **Cel:** gotowy adapter .bin, model opublikowany na HF Hub, opisana karta modelu.

CI + Docker — 4 h

- **Co robisz:** Tworzysz Dockerfile dla llama-infer (FastAPI + transformers + LoRA loader), Ustawiasz GitHub Actions: build obrazu przy pushu, push do GitHub Container Registry (ghcr.io), oznaczanie wersji (llama-infer:latest, llama-infer:qa-v1).
- **Cel:** działający workflow CI, obraz ghcr.io/user/llama-infer:qa-v1 publicznie dostępny z linkiem w README.

Kamień milowy III. miesiąca

Trzeci miesiąc to moment, w którym Twój kod zaczyna uczyć się na własnych danych. Masz za sobą pierwszy klasyczny training loop w PyTorchu, własny fine-tuning modelu językowego i integrację z Hugging Face. Twoje środowisko działa na GPU, loguje metryki i zaczyna przypominać prawdziwy workflow inżyniera AI.

Repozytorium *genai-playground* zawiera:

- train_mnist.py (własny training loop), cuda_check.ipynb, hf_infer.py,
- folder model/ (fine-tuned BERT na Pol-Emo) oraz lora_adapter/ (LoRA na tym samym zbiorze),
- działający dashboard Weights & Biases (link w README.md),

- plik journal/miesiac-3.md z podsumowaniem i metrykami,
- (opcjonalnie):
  – rotary_vs_baseline.md + journal/rotary_pe.md,
  – folder tinygpt_model/ z checkpointem tinygpt_model.pt,
  – folder llama_finetune/ z kartą modelu i linkiem do HF Hub,
  – obraz llama-infer:latest opublikowany w GHCR, workflow CI w .github/workflows.

Ten kamień milowy oznacza, że masz pierwsze własne modele, własny kod trenowania i własne logi – wszystko zintegrowane i gotowe do rozwijania w kolejnych miesiącach. Model działa, środowisko loguje, repo żyje.

Miesiąc IV — Prompt Engineering & Ewaluacja LLM

1. Prompt Patterns od podstaw — 4 h

- **Co robisz:** przerabiasz wzorce zero-/one-/few-shot, chain-of-thought, role-based. Tworzysz plik prompts_patterns.md z 10 przykładami (PL + EN).
- **Materiały:** „Prompt Engineering Guide" (sekcje Patterns & Anti-Patterns).
- **Cel:** każdy przykład przechodzi test «sens ≈ 95 %» wg własnej oceny.

2. RAG 101: Faiss + LangChain — 6 h

- **Co robisz:** budujesz mini-chat na dokumentach PDF (3 pliki), pipeline: chunking → embeddings → Faiss → LLM.
- **Materiały:** tutorial LangChain „Chat with your docs".
- **Cel:** zapytanie „Czego dotyczy drugi pdf?" zwraca poprawną odpowiedź.

3. Function-Calling / Tools (lokalny mock) — 5 h

- **Co robisz:** tworzysz prosty serwis tools_agent.py, definiujesz funkcję get_weather(city), LLM wybiera funkcję i zwraca JSON.
- **Materiały:** docs OpenAI Tools + repo openai-function-calling-python.
- **Cel:** log JSON-ów w konsoli + opis w README.

4. Suite testów: MMLU & TruthfulQA — 5 h

- **Co robisz:** instalujesz lm-eval-harness, uruchamiasz na modelu DistilGPT-2 oraz fine-tuned BERT z M-III.
- **Cel:** plik eval_results.json + porównanie metryk w journal/.

5. Cookbook promptów — 4 h

- **Co robisz:** kompilujesz dotychczasowe prompty do prompt_cookbook.md (sekcje: tekst, kod, obraz).
- **Cel:** minimum 20 opisanych promptów + tagi (classification, creative, debug).

6. Skrypty ewaluacyjne — 6 h

- **Co robisz:** eval.py liczy BLEU, BERTScore, a dla rozmów – własny rule-based LLM judge (gpt-3.5-turbo z metapromptem).
- **Cel:** jeden plik CSV z wynikami trzech metod dla 30 promptów.

7. Retrospektywa & journal — 2 h

- **Co robisz:** journal/miesiac-4.md — TOP 3 insighty nt. promptów i plany na next month.
- **Cel:** commit “M-IV: prompty + ewaluacja”.

Jeśli masz ekstra wolny weekend (8h)

Synthetic-Data Pipeline (Self-Instruct) — 8 h

- **Co robisz:** Budujesz prosty generator danych oparty na technice Self-Instruct: self_instruct.py generuje 300 par {"question": "...", "answer": "..."} z wybranego obszaru tematycznego (np. geografia, UX, prawo pracy), używasz gpt-3.5-turbo lub lokalnego modelu z LoRA do samodzielnego tworzenia QA. Weryfikujesz 10 przykładów ręcznie oraz testujesz 10 losowych par przez inny model (DistilGPT-2 lub BERT), oceniając jakość odpowiedzi.
- **Materiały:** Artykuł: Self-Instruct: Aligning Language Models with Self-Generated Instructions, Repo: self-instruct (https://github.com/yizhongw/self-instruct), Format danych: .jsonl, jeden przykład na linię
- **Cel:** gotowy plik data/self_instruct_qa.jsonl, ręczna walidacja 10 przykładów, zapis wyników w notes/self_instruct_eval.md.

Jeśli masz 2 ekstra wolne weekendy (16h)

RLHF demo z TRL-X (PPO) — 16 h

- **Co robisz:** Budujesz Reward Model (RM): dane: 500 przykładów {"prompt": "...", "good_answer": "...", "bad_answer": "..."}; model: BERT lub DistilBERT, zadanie: klasyfikacja preferencji (binary: good > bad), zapis: rm_model/, log loss i accuracy;

Następnie trenujesz DistilGPT-2 z PPO (Proximal Policy Optimization) na tych samych promptach, używając TRL-X (trl.train_rlhf).
Logujesz: średni RM-score przed i po treningu, ilość nagród dodatnich vs ujemnych.
Notatnik rlhf_demo.ipynb dokumentuje cały proces: wykres RM-score vs epoka, przykładowe odpowiedzi modelu przed/po, wnioski.

- **Materiały:** Repo: trlx (https://github.com/CarperAI/trlx), Artykuł: Deep RL from Human Preferences, Dataset przykładowy: Anthropic HH-RLHF, OpenAssistant/oasst1 (subset)
- **Cel:** folder rlhf_demo/ z rm_model/, ppo_model/, rlhf_demo.ipynb, średni RM-score po fine-tuningu > niż przed, opisany trend i wynik w notatniku.

Kamień milowy IV. miesiąca

Czwarty miesiąc to moment, w którym zaczynamy świadomie używać modeli – nie tylko generujemy odpowiedzi, ale też planujemy strukturę promptu, podpinamy własne dane i sprawdzamy, czy wynik ma sens.
Poznaliśmy wzorce promptowania, zbudowaliśmy nasz pierwszy system RAG i uruchomiliśmy benchmarki, które pozwalają porównać różne modele. To praktyczne minimum, od którego zaczyna się budowanie aplikacji opartych na LLM-ach.

Repozytorium *genai-playground* zawiera:

- pliki prompts_patterns.md i prompt_cookbook.md z opisanymi i przetestowanymi ≥ 20 promptami,
- działający mini-RAG rag_chat.py oparty na własnym korpusie, z podglądem działania w postaci animacji GIF (demo_rag.gif lub podobny),
- plik eval_results.json z wynikami lm-eval-harness dla 1–2 modeli (np. DistilGPT-2, BERT),
- skrypt eval.py, który przelicza BLEU, BERTScore lub własny scoring LLM → wyniki w scores.csv,
- wpis journal/miesiac-4.md z analizą promptów, wyników i planem na kolejne eksperymenty,
- (opcjonalnie) folder rlhf_demo/ lub self_instruct/ zawierający dodatkowe eksperymenty z RLHF lub syntetycznymi danymi QA, jeśli zostały zrealizowane zadania weekendowe.

Kamień milowy potwierdza, że potrafisz projektować i testować prompty, łączyć modele z własnymi danymi oraz obiektywnie oceniać ich działanie. To kluczowa umiejętność przy tworzeniu chatbotów, copilotów, systemów rekomendacji i interaktywnych aplikacji Gen-AI.

Miesiąc V — Multimodalność: Text ↔ Image / Audio / Code

W piątym miesiącu wychodzimy poza tekst i wchodzimy w świat multimodalności (nie pomyl z multimedialnością!). Nauczysz się generować obrazy za pomocą modeli diffusion, transkrybować i tworzyć dźwięk oraz korzystać z modeli językowych do kodu. Poznasz też podstawy integracji tych kanałów w jednej aplikacji.

1. Stable Diffusion XL — inferencja & podstawy promptów — 4 h

- **Co robisz:** uruchamiasz skrypt sdxl_infer.py, generujesz 10 obrazów z różnymi CFG i seed.
- **Materiały:** repo diffusers/examples/inference.
- **Cel:** folder outputs/sdxl/ + log promptów w prompt_gallery.csv.

2. ControlNet / LoRA-image fine-tune — 6 h

- **Co robisz:** fine-tunujesz model stylu (rank 4, 500 iter.) na 10 własnych zdjęciach; implementujesz load_lora_image.py.
- **Cel:** plik .safetensors < 200 MB + 3 przykładowe inferencje „before/after".

3. Whisper + AudioCraft — 6 h

- **Co robisz:** whisper_transcribe.py – transkrypt 5-min nagrania PL/EN, music_gen.py – 30-sek sample "lo-fi AI study beat".
- **Cel:** katalog audio/ z .wav + transkrypcja .txt.

4. Code-LLM quick-start — 6 h

- **Co robisz:** instalujesz StarCoder lub CodeQwen, generujesz patch do własnego sdxl_infer.py.
- **Materiały:** HF generate.py --instruction.
- **Cel:** commit z patchem wygenerowanym i opis w PR.

5. Mini-portfolio demos — 5 h

- **Co robisz:** tworzysz streamlit_app.py, prezentujesz: Tab 1 – text-to-image, Tab 2 – speech-to-text, Tab 3 – code-LLM autocomplete.
- **Cel:** lokalny URL + screen w README.

6. Model Cards & licencje — 3 h

- **Co robisz:** piszesz model card dla LoRA-image; sprawdzasz licencję SDXL/Whisper i dodajesz sekcję compliance.
- **Cel:** plik model_card.md + LICENSE w folderze modelu.

7. Retrospektywa & journal — 2 h

- **Co robisz:** journal/miesiac-5.md – co najbardziej Cię zaskoczyło w multimodalności?
- **Cel:** commit „M-V: multimodal ready".

Jeśli masz ekstra wolny weekend (8h)

DreamBooth LoRA na własną osobę — 8 h

- **Co robisz:** Zbierasz 15–20 zdjęć swojej twarzy w różnych warunkach i zapisujesz w katalogu training_data/. Przygotowujesz plik .caption z promptami w stylu:
  – photo of sks-person,
  – portrait of sks-person wearing glasses.
  Używasz interfejsu kohya_ss (lokalnie lub Colab), konfigurujesz trening LoRA na modelu stabilityai/sd-1.5 z użyciem --network_module=lycoris. Po treningu zapisujesz adapter .safetensors, wrzucasz na HF Hub jako repozytorium prywatne.
- **Materiały:**
  - kohya_ss GUI (https://github.com/bmaltais/kohya_ss);
  - Blog: DreamBooth LoRA in 2024 – how to personalize Stable Diffusion safely;
  - Hugging Face docs: Uploading files to private repo.

- **Cel:** folder lora_self/ z plikiem .safetensors, logami i linkiem do HF Hub (private), oraz min. 3 wygenerowanymi obrazami z promptu "sks-person in cyberpunk outfit" lub podobnego.

Jeśli masz 2 ekstra wolne weekendy (16h)

Multimodal chatbot (LLaVA-style) — 16 h

- **Co robisz:** Tworzysz prototyp chatbota, który:
  – przyjmuje zdjęcie od użytkownika (upload lub drag&drop),
  – generuje opis (caption) przy użyciu modelu CLIP lub BLIP,
  – przekazuje ten opis do LLM (np. vicuna-7b-v1.5 lub llama-3-8b-instruct) w promptcie typu "Based on this image: [CAPTION], answer: [QUESTION]".
  Budujesz demo w Streamlit lub Gradio. Opakowujesz wszystko w docker-compose.yml:
  – serwis front (Streamlit),
  – serwis caption-api,
  – serwis llm-api (z adapterem lub lokalnym inferencem).
- **Materiały:**
  – LLaVA paper + Hugging Face space demo,
  – GitHub: open-mmlab/mmgpt lub LLaVA-playground,
  – Blog: How to build your own vision-LLM pipeline.
- **Cel:** Folder chatbot_multimodal/ z kodem, działający docker-compose up lokalnie, plik README.md z opisem architektury i przykładowym zrzutem ekranu interfejsu. (opcjonalnie) nagranie demo (GIF lub mp4 ≤ 60 s).

Kamień milowy V. miesiąca

Kamień milowy potwierdza, że opanowałeś_aś generację i przetwarzanie trzech modalności, umiesz wytrenować LoRA i pokazać je w jednym interfejsie – fundament potrzebny do budowy nowoczesnych, wielomodalnych produktów Gen-AI.

Repozytorium *genai-playground* zawiera:

- sdxl_infer.py, controlnet_lora/, whisper_transcribe.py, music_gen.py,
- streamlit_app.py (demo 3-zakładkowe) + screenshot,
- model_card.md + LICENSE,

- journal/miesiac-5.md,
- (opcjonalnie) dreambooth_lora/ lub llava_chatbot/ z instrukcją uruchomienia.

Miesiąc VI — RAG-na-Sterydach & Agents

Nadszedł czas, by Twój model przestał być samotną wyspą i połączył się ze światem zewnętrznym. Chcemy, by odpowiadał na konkretne pytania o Twoje dane i wykonywał zadania krok po kroku. W tym miesiącu nauczymy się budować system RAG (Retrieval-Augmented Generation), porównać popularne bazy wektorowe i tworzyć własnych agentów, którzy myślą, planują i działają w Twoim imieniu.

Na koniec miesiąca będziesz mieć gotowy system Q&A oparty na własnych plikach, sprawdzone rozwiązania do chunkowania i wyszukiwania oraz prostą strukturę agentową, którą można rozwijać w kierunku chatbotów, copilotów czy automatycznych analityków.

Ps. Pamiętaj, by ustawić na swoim repo licencję AGPL-3.0 - to najmocniejsza ochrona z tych dostępnych na GitHubie. A od teraz Twój kod staje się naprawdę cenny.

1. Vector DB battle — Chroma vs Qdrant vs Weaviate — 6 h

- **Co robisz:** Instalujesz trzy silniki (Docker Compose), ładujesz ten sam 100-MB zbiór markdownów. Mierzysz: czas indeksacji, rozmiar na dysku, średni czas zapytania (k-NN = 3).
- **Materiały:** docs Chroma, Qdrant „Quickstart", Weaviate „Get started in 5 min".
- **Cel:** tabelka vector_db_benchmark.md z trzema metrykami.

2. Chunking & Embeddings tuning — 4 h

- **Co robisz:** implementujesz chunker.py (RecursiveCharacter + sentence-split) oraz testujesz dwa embeddery: text-embedding-3-small i all-MiniLM-L6-v2.
- **Cel:** notebook chunking_eval.ipynb pokazuje recall@3 dla obu embedderów.

3. LangChain RAG baseline — 5 h

- **Co robisz:** skrypt rag_chain.py buduje prostą aplikację Q&A (embeddings → retriever → GPT-4o).
- **Cel:** zapytanie „Jakie są kroki instalacji Qdrant?" zwraca poprawną odpowiedź + źródła.

4. Agents with LangGraph / CrewAI — 7 h

- **Co robisz:** tworzysz workflow z dwoma agentami: Retriever i Answer Refiner. Graph steruje kolejnością i kryje pętlę „refine until CE-score > 0.85".
- **Cel:** plik agents_flow.py; log pokazuje maks. 3 iteracje.

5. Guardrails & JSON-schema validation — 4 h

- **Co robisz:** nakładasz Guardrails (rail.json) na odpowiedzi: wymuszony JSON { "answer": str, "sources": [str] }, długość ≤ 2000 znaków.
- **Cel:** test pytest test_guardrails.py przechodzi 5 przykładów.

6. Load-test & latency profiling — 4 h

- **Co robisz:** używasz Locust (locustfile.py) – 50 równoległych użytkowników, 200 zapytań. Profilujesz czas i zużycie VRAM.
- **Cel:** raport locust_report.html + wpis w journal/.

7. Retrospektywa & journal — 2 h

- **Co robisz:** journal/miesiac-6.md — wnioski z bitwy DB i check-lista pain-pointów.
- **Cel:** commit „M-VI: RAG + Agents".

Jeśli masz ekstra wolny weekend (8h)

Memory-augmented Agent (ReAct + long-term store) — 8 h

- **Co robisz:** Rozszerzasz istniejącego agenta (LangGraph lub CrewAI) o długoterminową pamięć opartą o Chroma.
  Implementujesz prosty mechanizm przechowywania istotnych fragmentów rozmowy (store["session_id"] = memory_vector) oraz retrieval przy kolejnych pytaniach.
  Tworzysz prompt w stylu ReAct (Reasoning + Acting), który odwołuje się do poprzednich odpowiedzi: „Pamiętaj, że wcześniej rozmawialiśmy o X…".
  Logujesz 2–3 sesje rozmowy z agentem, w których agent poprawnie używa pamięci.
- **Materiały:**
  – Paper: ReAct: Synergizing Reasoning and Acting in Language Models,
  – LangChain Memory docs,
  – Repos: langgraph/examples, CrewAI-memory.
- **Cel:**
  – skrypt memory_agent.py z aktywną pamięcią (Chroma + session-id),
  – plik chat_log.json z zapisaną rozmową i odwołaniami do pamięci,
  – plik notes/react_prompt.md z opisem działania promptu.

Jeśli masz 2 ekstra wolne weekendy (16h)

Cloud-scale RAG (AWS Bedrock lub Vertex AI) + semantic cache — 16 h

- **Co robisz:** Deployujesz retriever oparty na Qdrant Serverless (np. przez qdrant-cloud), ładujesz dane za pomocą API.
  Korzystasz z jednego z dużych modeli (Amazon Bedrock – Claude, Titan; lub Google Vertex AI – Gemini Pro), integrujesz go jako backend LLM.
  Wprowadzasz warstwę semantic cache z Redis:
  – przed wykonaniem zapytania sprawdzasz embedding zapytania,
  – jeśli podobieństwo > próg, zwracasz zapisany response z cache.
  Mierzysz oszczędności w liczbie żądań i czasie odpowiedzi.

- **Materiały:**
  – Qdrant Cloud docs + Bedrock/Vertex quickstarts,
  – Blog: Semantic Cache with Redis and OpenAI,
  – Artykuł: Optimizing LLM latency with caching
- **Cel:**
  – folder cloud_rag/ z plikiem cloud_rag_flow.md,
  – zapis benchmarku: cache-hit ratio, koszt zapytań z/bez cache,
  – screenshot dashboardu (np. Redis Insight / GCP log view).

Kamień milowy VI. miesiąca

Szósty miesiąc za nami – mamy system, który potrafi przeszukiwać dokumenty, odpowiadać na pytania i działać zgodnie z planem agenta. Poznaliśmy różne silniki wektorowe, umiemy kontrolować jakość chunków i zbudowaliśmy bazowego agenta LangGraph lub CrewAI.

Repozytorium *genai-playground* zawiera:

- vector_db_benchmark.md z porównaniem metryk Chroma, Qdrant i Weaviate (czas, rozmiar, recall),
- chunker.py + chunking_eval.ipynb z testami recall@3 i porównaniem embeddingów,
- działający skrypt rag_chain.py (Q&A z własnych dokumentów, z cytowanymi źródłami),
- agents_flow.py (LangGraph / CrewAI) z przykładowym logiem działania,
- rail.json z regułami Guardrails + test_guardrails.py,
- raport locust_report.html z testów obciążeniowych,
- journal/miesiac-6.md z podsumowaniem i checklistą,
- (opcjonalnie)
  – folder memory_agent/ z agentem pamięci i logiem sesji,
  – folder cloud_rag/ z opisem integracji z chmurą i benchmarkiem semantic-cache.

Kamień milowy VI potwierdza, że potrafisz zbudować, zoptymalizować i testować systemy RAG oraz agentowe workflow – czyli jedną z najczęściej wdrażanych architektur w nowoczesnych produktach Gen-AI.

Miesiąc VII — Optymalizacja modeli & Inference on Edge

W tym miesiącu skupimy się na optymalizacji. Dowiemy się, jak zmniejszyć rozmiar modelu, jak przyspieszyć inferencję i uruchomić model LLM lokalnie — nawet bez dostępu do GPU. To kluczowe umiejętności, jeśli chcesz tworzyć lekkie, tanie i dostępne aplikacje Gen-AI.

Poznamy też techniki kwantyzacji, łączenia adapterów LoRA, wykorzystamy llama.cpp, vLLM i TensorRT, a na koniec zbudujemy serwis FastAPI do lokalnej inferencji. Dzięki temu będziemy mogli porównać różne podejścia, oszacować koszty i przygotować rozwiązania działające również na słabszym sprzęcie – od laptopów po edge-devices.

1. Quantization basics (BitsAndBytes / GGUF) — 4 h

- **Co robisz:** instalujesz bitsandbytes, konwertujesz Llama-3-8B do INT8 i 4-bit GGUF (llama.cpp/convert.py).
- **Cel:** pliki llama-3-8b-int8.bin oraz gguf/llama-3-8b.q4_K_M.gguf; tabela rozmiarów w README.

2. LoRA fusion & MergeKit — 4 h

- **Co robisz:** łączysz bazowy Llama-3 z adapterem LoRA z M-III (mergekit.yaml, mergekit-cli).
- **Cel:** jeden scalony .safetensors < 6 GB + log merge w journal/.

3. llama-cpp-python serwer CPU — 4 h

- **Co robisz:** instalujesz llama-cpp-python, uruchamiasz serwis python server.py --model gguf/… --n_threads 8.
- **Cel:** endpoint http://localhost:8000/completion zwraca odpowiedź w < 1,5 s (prompt 20 tok).

4. Kompilacja TensorRT-LLM / vLLM — 6 h

- **Co robisz:** build obrazu tensorrt-llm:dev, zamieniasz model w format FP16, uruchamiasz benchmark. alternatywa: vllm + Flash-Attention 2.
- **Cel:** throughput (tok/s) ≥ 2× wersji PyTorch fp16.

5. Benchmark matrix script — 6 h

- **Co robisz:** bench.py zbiera: tryb (fp16, int8, q4), VRAM, latency, throughput. Zapisuje CSV + generuje wykres matplotlib „latency vs VRAM".
- **Cel:** plik bench_results.csv + latency_vram.png w reports/.

6. FastAPI / gRPC micro-service + Docker — 6 h

- **Co robisz:** app/main.py (FastAPI): /health, /chat (stream SSE); Dockerfile minimalny (~1,2 GB), build w CI; push do GHCR.
- **Cel:** docker run -p 8000:8000 llama-infer:edge → lokalny chat w przeglądarce.

7. Retrospektywa & journal — 2 h

- **Co robisz:** journal/miesiac-7.md — porównanie kosztów GPU-cloud vs CPU-edge; TODO na M-VIII.
- **Cel:** commit „M-VII: edge-optim ready".

Jeśli masz ekstra wolny weekend (8h)

On-device mobile (Android/iOS) z GGML — 8 h

- **Co robisz:** Eksportujesz wcześniej przygotowany model .gguf (np. Llama 3 8B quantized q4_K_M) do użycia w llama.cpp.
  Konfigurujesz lokalny build llama.cpp z backendem ggml lub metal (dla Apple).
  Tworzysz prostą aplikację mobilną z użyciem Expo / React Native lub Flutter, która:
  – umożliwia wpisanie promptu,

– wyświetla odpowiedź lokalnie wygenerowaną przez model.
Całość działa na urządzeniu z ≤ 2 GB RAM.

- **Materiały:**
  – llama.cpp mobile build instructions:
  https://github.com/ggerganov/llama.cpp/tree/master/examples
  – Flutter/Expo starter templates
  – Tutorial: Running LLMs on phones with llama.cpp
- **Cel:** folder mobile_ondevice/ z apką mobilną, dokumentacją instalacji i filmikiem z działania (GIF lub mp4 ≤ 30 sek).

Jeśli masz 2 ekstra wolne weekendy (16h)

Jetson / Raspberry Pi voice-agent — 16 h

- **Co robisz:** Instalujesz środowisko na Jetson Nano (CUDA 12) lub Raspberry Pi 5 (Linux ARM):
  – llama.cpp (z ggml),
  – whisper.cpp do przekształcenia mowy na tekst (STT),
  – coqui-tts lub espeak-ng do syntezy mowy (TTS).
  Budujesz pipeline: mikrofon (np. USB) nagrywa → STT (whisper) rozpoznaje zapytanie → LLM (llama) odpowiada → TTS odczytuje odpowiedź na głos.
  Całość działa w trybie offline.
- **Materiały:**
  – whisper.cpp: https://github.com/ggerganov/whisper.cpp
  – coqui-ai/TTS lub espeak-ng
  – Projekt: Voice Assistant on Raspberry Pi with Local LLMs
- **Cel:** folder jetson_voice/ z kodem pipeline'u, opisem konfiguracji i nagraniem audio z działania agenta.

Kamień milowy VII. miesiąca

W tym miesiącu nauczyliśmy się kompresować modele, obsługiwać je lokalnie i porównywać różne strategie inferencji. Zbudowaliśmy prosty serwis do obsługi LLM z użyciem llama-cpp, przygotowaliśmy benchmarki wydajnościowe i zautomatyzowaliśmy cały proces w kontenerze.

Repozytorium *genai-playground* zawiera:

- older quantization/ z modelami INT8 i .gguf, plus tabela porównawcza rozmiarów i jakość,
- mergekit/merged.safetensors + plik merge_log.txt,
- działający serwis server.py oparty na llama-cpp-python, gotowy do wdrożenia,
- Dockerfile + workflow build + uruchomienie w kontenerze,
- bench.py, bench_results.csv, latency_vram.png – pomiary czasu i użycia pamięci,
- journal/miesiac-7.md z wnioskami z testów,
- (opcjonalnie) folder mobile_ondevice/ lub jetson_voice/ – z działającym pipeline i dokumentacją.

Kamień milowy potwierdza, że potrafisz kompresować, scalać i serwować modele z minimalnym śladem pamięci, a także uruchamiać je poza środowiskiem chmurowym — np. na edge-device, w aplikacji mobilnej lub w Dockerze. To niezbędne umiejętności przy wdrażaniu modeli w realnych produktach.

Miesiąc VIII — MLOps Lite: CI/CD → Registry → Monitoring

W ósmym miesiącu nauczymy się automatyzować i kontrolować to, co zbudowaliśmy do tej pory. Skonfigurujemy testy i pipeline'y CI/CD, wdrożymy nasze modele w kontenerach i zaczniemy wersjonować je jak prawdziwe pakiety. Poznamy też podstawy monitorowania — tak, żeby wiedzieć, co dzieje się z naszym modelem po wdrożeniu i jak wcześnie wychwycić problemy.

To pierwszy krok w stronę lekkiego MLOps — wystarczająco prostego, by nie przytłoczyć, ale wystarczająco skutecznego, by wdrożenia nie były ręcznym chaosem. Dzięki temu nasze projekty staną się powtarzalne, czytelne i gotowe do skalowania.

1. GitHub Actions → test + build + push — 6 h

- **Co robisz:** workflow: pytest → docker build → push do GitHub Container Registry tag wg semver i sha modelu
- **Materiały:** tutorial "CI/CD with GitHub Actions & Docker" (freeCodeCamp)
- **Cel:** zielony ✔ workflow przy każdym push do main.

2. Model Registry = MLflow + HF Hub — 4 h

- **Co robisz:** mlflow server --backend-store-uri ./mlruns; logujesz parametry LoRA z maja, rejestrujesz wersję "v1.0.0"; automatyczny hf_hub_upload() po udanym runie.
- **Cel:** karta modelu na HF Hub + UI MLflow z dwoma runami.

3. KServe / vLLM InferenceService (kind) — 6 h

- **Co robisz:** lokalny klaster kind, instalacja KServe, manifest InferenceService z obrazem llama-infer:edge.
- **Materiały:** tutorial "KServe on kind"
- **Cel:** kubectl get inferenceservices → status Ready; curl /v1/chat/completions działa.

4. **Monitoring: Prometheus + Grafana — 5 h**
   - **Co robisz:** docker-compose prometheus.yml + grafana. Middleware prometheus_fastapi_instrumentator zbiera latency, tokens.
   - **Cel:** dashboard "LLM latency & throughput" z średnią i p99 sciezka-kariery-cv.
5. **Drift & Quality watcher (Evidently) — 5 h**
   - **Co robisz:** nightly job drift_check.py porównuje embedding-cosine rozkłady; alarm do Grafany przy PSI > 0.2.
   - **Cel:** widoczny panel "Data drift" z ostatnich 7 dni.
6. **Nightly LoRA retrain pipeline — 6 h**
   - **Co robisz:** GitHub Actions "schedule: cron '0 3 * * *'" → retrain na nowych 2 k promptów z kolejki.
   - **Cel:** automatyczny PR z nowym adapterem i tag nightly-YYYY-MM-DD.
7. **Retrospektywa & journal — 2 h**
   - **Co robisz:** journal/miesiac-8.md — co poszło lekko, co zajęło najdłużej, plan na M-IX.
   - **Cel:** commit "M-VIII: mlops-lite ready".

Jeśli masz ekstra wolny weekend (8h)

AWS SageMaker JumpStart deploy — 8 h

- **Co robisz:** Korzystamy z gotowego kontenera huggingface-pytorch-inference w SageMaker JumpStart.
  Tworzymy endpoint Async z modelem (np. distilbert-base-uncased lub własny adapter z HF Hub).
  Wysyłamy 50 testowych zapytań i generujemy raport:
  – czas odpowiedzi,
  – koszt działania (na podstawie CloudWatch i Billing).
- **Materiały:**
  – AWS doc: JumpStart Inference with Hugging Face models,

– Blog: Deploying LLMs to SageMaker Async Endpoints,
– HF: inference_toolkit GitHub examples.

- **Cel:** działający endpoint (Async), plik sagemaker_deploy.md z instrukcją, URL endpointu i podsumowaniem kosztu ($/1k tokenów).

Jeśli masz 2 ekstra wolne weekendy (16h)

GitOps + Argo CD blue-green — 12 h

- **Co robisz:** Tworzymy osobne repo infra/ z definicją manifestów (np. inference-v1.yaml, inference-v2.yaml).
  Konfigurujemy Argo CD do automatycznego śledzenia zmian w repo i wdrażania ich jako ArgoApp.
  Ustawiamy blue-green rollout: 10% ruchu do nowej wersji, stopniowe przejście do 100% przy braku błędów.

Chaos test — 4 h

- **Co robisz:** Wywołujemy kubectl delete pod <name> — symulujemy awarię. Sprawdzamy, czy:
  – Prometheus zgłasza alarm (latency ↑, token error ↑),
  – Grafana pokazuje wykres incydentu,
  – pod restartuje się automatycznie i system wraca do normy.
- **Materiały:**
  – Argo CD: GitOps tutorial with Helm and Kustomize,
  – Blog: Blue-Green Deployment with Kubernetes and Argo,
  – CV e-book: „GitOps i chaos engineering w AI" (rozdział 10)
- **Cel:** folder infra/ z konfiguracją rolloutu,
  dashboard z alarmem, log incydentu,
  wpis chaos_report.md z opisem testu i wynikami.

Kamień milowy VIII. miesiąca

Kolejny miesiąc za nami – nasze repo przeszło metamorfozę: z projektu lokalnego stało się systemem zautomatyzowanym, wersjonowanym i monitorowanym. Umiemy już wdrażać modele, budować pipeline'y, sprawdzać jakość danych i automatycznie uruchamiać retrain, kiedy to potrzebne. Jeśli

zrealizowaliśmy zadania dodatkowe, nasze modele działają już w chmurze i aktualizują się bez ręcznego klikania.

Repozytorium *genai-playground* zawiera:

- .github/workflows/ci.yml z etapami: lint → test → build + push obrazu do GHCR,
- folder mlruns/ z zarejestrowanym modelem + link do HF Hub,
- folder kserve/ z manifestem inference.yaml gotowym do wdrożenia,
- plik docker-compose.monitoring.yml + dashboard .json (Prometheus + Grafana),
- skrypt drift_check.py z zapisanym alarmem (np. PSI > 0.2) i powiadomieniem,
- workflow nightly-retrain.yml w trybie cron (np. codziennie 3:00),
- wpis journal/miesiac-8.md z analizą wyników i planem iteracji,
- (opcjonalnie):
  – folder sagemaker_deploy/ z opisem i testami endpointu,
  – folder infra/ z konfiguracją Argo CD, rolloutem i raportem z chaos testu.

Kamień milowy VIII potwierdza, że potrafisz zautomatyzować cały cykl: kod → build → deploy → monitor → retrain, i to bez potrzeby ręcznej ingerencji. To podstawa nie tylko dla rozwoju projektów AI, ale i dla ich skalowania w środowisku produkcyjnym.

Miesiąc IX — Responsible & Secure Gen-AI

W dziewiątym miesiącu skupimy się na odpowiedzialności. Nauczymy się sprawdzać, co nasz model może powiedzieć — i czego zdecydowanie nie powinien. Wprowadzimy mechanizmy filtrujące, przetestujemy modele pod kątem toksyczności, uprzedzeń i podatności na prompt injection. Przyjrzymy się też interpretowalności, ochronie danych osobowych oraz zgodności z unijnym AI Act.

To etap, w którym zaczynamy traktować nasz system poważnie — jak coś, co może wejść na produkcję i musi spełniać realne standardy jakości, bezpieczeństwa i prawa.

1. Content-Filters + Policy Check — 4 h

- **Co robisz:** implementujesz filtr OpenAI prompt-policy (treści zabronione, wrażliwe, reguły tonacji) w filters.py; integrujesz z FastAPI middleware (blokada lub „refuse to comply").
- **Cel:** 100 % promptów testowych z listy banned jest odrzucanych, log w filters.log.

2. Bias / Toxicity / Jailbreak Tests — 6 h

- **Co robisz:** uruchamiasz HarmBench i pakiet red-teaming promptinject. raportujesz metryki toxicity@95, jailbreak-success dla modelu Llama-3-8B.
- **Cel:** plik harm_report.html + CSV z wynikami.

3. Interpretability — logits-lens & AttentionViz — 6 h

- **Co robisz:** notebook interpret_logits.ipynb – warstwa-po-warstwie predykcje tokenów; attention_viz.py rysuje heat-mapę atencji dla 3 przykładów.
- **Cel:** folder reports/explain/ z grafikami + krótki opis w README.

4. GDPR / PII Redaction Pipeline — 4 h

- **Co robisz:** używasz presidio lub pii-scrubber do anonimizacji zapytań/odpowiedzi; testujesz na 20 zdaniach zawierających imię, PESEL, adres.
- **Cel:** precision ≥ 0,95, recall ≥ 0,90; log w pii_stats.json.

5. Watermarking & Provenance — 4 h

- **Co robisz:** wdrażasz niewidoczny watermark (DENDI / QuIP) w generowanym tekście, skrypt detect_watermark.py osiąga accuracy > 0,9 na 100 próbkach.
- **Cel:** plik watermark_demo.md z opisem + wykres ROC.

6. EU AI Act Mini-Assessment & Model/Data Card — 6 h

- **Co robisz:** wypełniasz checklistę wymogów (system ograniczonego ryzyka); aktualizujesz model_card.md sekcje Fairness, Safety, EU AI Act.
- **Cel:** ai_act_assessment.pdf wygenerowany z szablonu + link w README.

7. Retrospektywa & journal — 2 h

- **Co robisz:** journal/miesiac-9.md — największy problem, plan mitigacji, TODO na M-X.
- **Cel:** commit “M-IX: responsible-ai ready”.

Jeśli masz ekstra wolny weekend (8h)

Red-teaming sprint + mitigacje — 8h

- **Co robisz:** Tworzymy zbiór 100 promptów próbujących obejść ograniczenia modelu (prompt injection, jailbreak, niejawna przemoc, dezinformacja, instrukcje niezgodne z polityką).
  Uruchamiamy testy z filters.py, zapisujemy filters.log, analizujemy skuteczność (ile promptów zostało prawidłowo odrzuconych).
  Poprawiamy prompt-guard: rozszerzamy blacklistę, zmieniamy prompt systemowy, wprowadzamy prostą logikę heurystyczną (np. analiza słów kluczowych).
  Powtarzamy test do momentu osiągnięcia ≥ 90% skuteczności blokady.

- **Materiały:**
  – Paper: Universal and Transferable Adversarial Attacks on Aligned Language Models (2023),
  – Narzędzia: promptinject, harmbench, własne testy,
  – Blog: How to build an LLM firewall.
- **Cel:** folder redteaming_weekend/ z plikiem attack_prompts.jsonl, filters.log i mitigation_notes.md z opisem zmian i wynikami testu.

Jeśli masz 2 ekstra wolne weekendy (16h)

DP-SGD tiny-LLM (Opacus) + epsilon report — 16 h

- **Co robisz:** Przygotowujemy mały korpus tekstów zawierających dane wrażliwe (np. syntetyczne CV, maile, numery PESEL).
  Trenujemy model DistilGPT-2 z użyciem Opacus i różnicowania gradientów (dp_optimizer + PrivacyEngine).
  Ustawiamy parametry tak, by końcowe $\varepsilon \leq 8$ (dla $\delta = 1e\text{-}5$).
  Równolegle trenujemy model referencyjny (bez DP).
  Porównujemy perplexity, loss i zachowanie modelu (np. czy „ucieka" z danych wrażliwych).
- **Materiały:**
  – Opacus docs: https://opacus.ai/docs,
  – Paper: Training language models with differential privacy (2021),
  – Blog: What is ε in DP and why it matters.
- **Cel:** folder dp_sgd_model/ zawiera:
  trenowany model,
  plik dp_train_log.txt,
  dp_report.md z tabelą: ε, PPL, komentarze jakościowe.

Kamień milowy IX. miesiąca

W tym miesiącu nasz model przeszedł swój pierwszy „audyt". Przetestowaliśmy go pod kątem toksyczności, błędów bezpieczeństwa, przecieków danych i zgodności z przepisami. Potrafimy też ograniczyć ryzyka: wdrożyć filtry, redakcję danych i zabezpieczenia.

Repozytorium *genai-playground* zawiera:

- filters.py + filters.log (wszystkie „banned prompts" skutecznie odrzucone),
- harm_report.html + harm_results.csv z testów HarmBench lub własnych,
- folder reports/explain/ z grafikami: logits-lens, attention heatmap,

- folder pii_redaction/ z pipeline’em + metryki pii_stats.json,
- plik watermark_demo.md + skrypt detect_watermark.py,
- ocena ai_act_assessment.pdf + uzupełniony model_card.md (Fairness, Safety, Legal),
- wpis journal/miesiac-9.md z refleksją i checklistą ryzyk,
- (opcjonalnie):
  – folder redteaming_weekend/ z promptami atakującymi i poprawionym guardem,
  – folder dp_sgd_model/ z raportem prywatności i wynikami modeli.

Kamień milowy IX potwierdza, że potrafisz testować, zabezpieczać i dokumentować modele Gen-AI zgodnie z wymogami etycznymi i prawnymi — czyli możesz spokojnie iść na rozmowę z działem bezpieczeństwa lub audytem regulacyjnym.

Miesiąc X — Gen-AI w produkcie: Frontend × UX × Growth

W tym miesiącu chcemy zmienić model w kompleksowy produkt. Nauczymy się projektować interfejsy użytkownika, integrować frontend z API LLM, streamować odpowiedzi w czasie rzeczywistym i kontrolować dostęp do funkcji. Dodamy też płatności, A/B testy oraz flagi funkcjonalności – wszystko po to, żeby nasza aplikacja była używalna, skalowalna i gotowa na realnych użytkowników.

1. Wzorce UX dla Gen-AI — chat, copilot, sidebar, canvas — 4 h

- **Co robisz:** analizujesz 10 popularnych interfejsów (ChatGPT, Notion AI, Figma AI…) i zapisujesz checklistę komponentów: kontekst, historia, przycisk „regenerate", loader tok-po-toku.
- **Cel:** plik ux_patterns.md z tabelą plus screenshoty.

2. Frontend Next.js + shadcn/ui + LangChain.js — 6 h

- **Co robisz:** tworzysz projekt next-genai-ui; implementujesz stronę z czatem (Textarea + Button + MessageList), Tailwind config = paleta z design-systemu ścieżek CV.
- **Cel:** npm run dev → http://localhost:3000/chat działa i ma dark/light switch.

3. Streaming odpowiedzi (SSE/WebSocket) — 4 h

- **Co robisz:** podmieniasz fetch na strumień Server-Sent Events z FastAPI (/chat/stream); pokazujesz animowany „…" gdy przychodzą tokeny.
- **Cel:** średnia latencja pierwszego tokenu < 2 s; film GIF w README.

4. Middleware: Billing & Rate-limit — 4 h

- **Co robisz:** Express middleware ratelimit.ts (Redis) – 100 zapytań/dzień. stripe-checkout doładowuje 5 000 tok = 5 €.
- **Cel:** statyczna strona /pricing + webhook Stripe zapisuje tokeny do DB.

5. A/B test & Feature Flags (GrowthBook) — 4 h

- **Co robisz:** dodajesz flagę fast_llm (vLLM vs OpenAI). 50 % ruchu trafia na każdą opcję, eventy logowane JS SDK.
- **Cel:** dashboard pokazuje różnicę p95 latency.

6. Case-study: AI-Copilot for Docs — 8 h

- **Co robisz:** pobierasz dokumentację API (OpenAPI), ładujesz do wektor-DB, UI: wyszukiwarka + sekcja „insert snippet" do edytora Markdown.
- **Cel:** demo wideo (≤ 120 s) i link do działającej wersji.

7. Retrospektywa & journal — 2 h

- **Co robisz:** journal/miesiac-10.md — wnioski z A/B, lista poprawek UX.
- **Cel:** commit „M-X: product-ux ready".

Jeśli masz ekstra wolny weekend (8h)

Voice-interface (Web Speech API + Whisper) — 8 h

- **Co robisz:** Dodajesz do swojego frontendu (Next.js / Streamlit) przycisk 🎙️, który uruchamia Web Speech API do nagrywania głosu użytkownika.
  Zapisany audio-input (np. .webm lub .wav) przesyłasz do backendu, gdzie konwertujesz go na tekst za pomocą Whisper (openai/whisper, whisper.cpp lub faster-whisper).
  Tekst trafia do LLM (Llama, OpenAI, Mistral), a odpowiedź zamieniasz z powrotem na mowę za pomocą Coqui-TTS lub pyttsx3.
  Całość działa dla języka polskiego i angielskiego.
- **Materiały:**
  – Web Speech API:
  https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API
  – Whisper: faster-whisper GitHub,
  – TTS: https://github.com/coqui-ai/TTS
- **Cel:** folder voice_interface/ z kodem front/backend,
  demo z interakcją głosową (mp4 lub .gif),
  wpis README.md z instrukcją i przykładami zapytań.

Jeśli masz 2 ekstra wolne weekendy (16h)

Multi-tenant SaaS demo (Stripe + Postgres RLS) — 16 h

- **Co robisz:** Projektujesz bazę danych z tabelami:
  – orgs (firmy/zespoły),
  – users (z loginem i rolą),
  – usage (tokeny / zapytania / czas).
  Konfigurujesz Row-Level Security (RLS) w PostgreSQL tak, by użytkownicy widzieli tylko swoje dane (przez current_user_id).
  Budujesz panel admina z listą subskrypcji, płatności (Stripe), oraz wykresem zużycia tokenów (Chart.js lub Recharts).
  Wprowadzisz obsługę planów płatniczych (Free, Pro, Enterprise) i dashboard z miesięcznym usage.
- **Materiały:**
  – Stripe docs: SaaS Billing,
  – PostgreSQL: Using Row-Level Security,
  – Blog: Multi-tenant apps with Next.js and Supabase/Postgres.
- **Cel:** folder saas_multitenant/ z kodem backend + panelu admina, screeny z wykresami i bazą danych, testowe konto „admin" z widokiem subskrypcji i usage.

Kamień milowy X. miesiąca

W tym miesiącu udało nam się przekształcić model LLM w gotowy produkt. Zbudowaliśmy interfejs użytkownika z dobrym UX, wdrożyliśmy streaming tokenów, wersjonowanie funkcji i ograniczenia API.

Repozytorium *genai-playground* zawiera:

- ux_patterns.md + zrzuty popularnych interfejsów (chat, copilot, canvas),
- projekt next-genai-ui/ z czatem, streamingiem SSE i dark/light mode,
- middleware ratelimit.ts + działającą integrację Stripe Checkout,
- dashboard A/B testów w GrowthBook (fast_llm ON/OFF), zrzut PDF,
- folder docs_copilot/ – Q&A z dokumentacją API, demo (mp4 lub GIF),
- wpis journal/miesiac-10.md z analizą UX i planem zmian,
- (opcjonalnie):
  – folder voice_interface/ z interfejsem głosowym,
  – folder saas_multitenant/ z bazą danych, RLS i widokiem admina.

Kamień milowy dowodzi, że potrafimy opakować model Gen-AI w realny produkt SaaS – gotowy do testów z użytkownikami, do wdrożenia w firmie lub zaprezentowania inwestorowi.

Miesiąc XI — Portfolio, Community & Open-Source

W przedostatnim miesiącu naszej ścieżki kariery skupimy się na tym, by wszystko, co do tej pory zbudowaliśmy, miało swoje miejsce, porządek i wersję. Zadbamy o dokumentację, refaktoryzację, opiszemy nasze modele i zadbamy o widoczność w sieci.

Pokażemy też, że potrafimy pracować w zespole: dołożymy swoją pierwszą cegiełkę do istniejącego projektu open-source, przygotujemy demo, opowiemy o naszej ścieżce lub projekcie w formie wpisu albo nagrania. To miesiąc, w którym zaczynamy być widoczni jako twórcy, nie tylko użytkownicy modeli.

1. Refactor & release repo v1.0.0 — 4 h

- **Co robisz:** porządkujesz strukturę genai-playground (monorepo → apps/, libs/, infra/), odpalasz semantic-release.
- **Cel:** tag Git v1.0.0 + changelog; wszystkie testy zielone w CI.

2. Dokumentacja publiczna (Docusaurus) — 4 h

- **Co robisz:** instalujesz Docusaurus, generujesz docs API (FastAPI → OpenAPI → redocly-cli), sekcja „Quick Start" + „FAQ".
- **Materiały:** tutorial „Docs as Code" z e-booka CV sciezka-kariery-cv.
- **Cel:** docs.genai-playground.local pokazuje stronę startową z dark/light.

3. Kontrybucja OSS (Transformers / LangChain / Diffusers) — 6 h

- **Co robisz:** wybierasz issue good first issue, forkujesz, PR: poprawka docs lub testu.
- **Cel:** merged PR i plakietka w profilu GitHub „1st PR merged".

4. Blog / video tutorial „How I built my Gen-AI stack" — 5 h

- **Co robisz:** wpis na dev.to lub 8-min film na YouTube; pokazujesz demo, kod i wyniki benchmarku.
- **Cel:** min. 200 odsłon lub 20 👍 w 7 dni; link w README.

5. Mock interviews + LeetCode set — 5 h

- **Co robisz:** 3 sesje peer-to-peer (Pramp / Interviewing.io) + 30 pytań LeetCode medium (Python).
- **Cel:** feedback ≥ "Strong Hire" w ≥ 2 sesjach; screenshot opinii w journal/.

6. CV + LinkedIn upgrade (Gen-AI KPI) — 4 h

- **Co robisz:** dodajesz projekty, metryki (tok/s, koszt ↓ %, F1 ↑ %), linki demo.
- **Cel:** PDF CV w portfolio/cv_genai.pdf, LinkedIn "provides services: Generative AI".

7. Retrospektywa & journal — 2 h

- **Co robisz:** journal/miesiac-11.md — najcenniejsza lekcja z OSS, plan na M-XII.
- **Cel:** commit „M-XI: portfolio ready".

Jeśli masz ekstra wolny weekend (8h)

Lightning-talk @ meetup (nagranie) — 8 h

- **Co robisz:** Przygotowujesz krótki, zwięzły talk (5 slajdów / ~6 minut) pod tytułem „Edge-LLM w 1 GB RAM" lub inny temat techniczny z Twojego projektu. Prezentacja może być nagrana:
  – jako screencast z komentarzem,
  – telefonem / kamerą z podpiętym slajdem.
  Nagraj, zmontuj (opcjonalnie) i wrzuć na YouTube jako niepubliczny lub publiczny film z opisem i linkiem do repo.
  Wklej link do README.md lub osobnego pliku lightning_talk.md.
- **Materiały:**
  – Talk template: https://slides.com lub Canva Presentations,
  – Blog: How to give your first tech talk (remotely)
  – Inspiracja: lightning talki z PyData / Hugging Face Meetups.
- **Cel:** folder lightning_talk/ z prezentacją (PDF lub link), nagraniem na YT i krótkim streszczeniem treści.

Jeśli masz 2 ekstra wolne weekendy (16h)

Gen-AI Debug Toolkit (CLI + notebook) — 16 h

- **Co robisz:** Budujemy narzędzie do testowania i debugowania promptów oraz modeli LLM w lokalnym środowisku. Funkcjonalności w wersji 1.0:
  – debug_prompt.py: narzędzie CLI pozwalające wysłać prompt do wybranego modelu (OpenAI, Mistral, llama.cpp) i otrzymać:
  –– długość promptu w tokenach,
  –– estymowany koszt i czas odpowiedzi,
  –– scoring toksyczności / halucynacji / JSON-validity (heurystyczny),
  –– porównanie odpowiedzi z kilkoma wariantami promptu.
  – PromptDebugToolkit.ipynb: interaktywny notatnik z GUI (Gradio lub IPyWidgets), który pozwala testować prompty z różnymi modelami i parametrami.
  Repo zawiera requirements.txt, README.md, folder samples/ z przykładowymi przypadkami testowymi i tests/ z testami jednostkowymi.
- **Materiały:**
  – OpenAI / Hugging Face / Llama.cpp API wrappers
  – tiktoken, toxicity, jsonschema, evaluate (Hugging Face)
  – Blogi:
  –– How to build a prompt benchmarking tool,
  –– Debugging LLM behavior at scale (Anthropic),
  –– Inspiracja: lm-eval-harness, promptfoo, llm-debugger
- **Cel:** folder genai_debug/ zawierający:
  – debug_prompt.py (CLI) i PromptDebugToolkit.ipynb (notebook),
  – przykładowe wyniki testów (sample_output.jsonl),
  – README.md z instrukcją użycia i przykładami.
  Narzędzie nadaje się do użytku w Twoim capstone project (Miesiąc XII) lub jako osobny projekt open-source.

Kamień milowy XI. miesiąca

W tym miesiącu doprowadziliśmy projekt do wersji 1.0. Zrobiliśmy porządek w kodzie, zadbaliśmy o dokumentację i przygotowaliśmy portfolio. Ot, takie wiosenne przedświąteczne porządki, zanim teściowa przyjedzie - żeby wszystko wyglądało jak należy.

Repozytorium *genai-playground* zawiera:

- tag v1.0.0 + changelog (CHANGELOG.md),
- folder docs/ z dokumentacją projektu (np. Docusaurus) i działającą wersją hostowaną,
- link do zaakceptowanego pull requestu w projekcie open-source (transformers, langchain, diffusers, itp.),
- wpis blogowy lub nagranie wideo (YouTube/dev.to), wraz ze statystykami odsłon lub reakcji,
- folder portfolio/ z plikiem cv_genai.pdf i screenshotami z mock-interview,
- wpis journal/miesiac-11.md z podsumowaniem i planem na miesiąc XII,
- (opcjonalnie):
  – folder lightning_talk/ z prezentacją i nagraniem,
  – folder genai_debug/ z działającym narzędziem do testowania promptów i modeli LLM, dokumentacją i przykładowym flow.

Kamień milowy XI potwierdza, że potrafisz nie tylko budować, ale też dokumentować, porządkować i prezentować efekty swojej pracy. Masz gotowe, publiczne repo, pierwszą kontrybucję open-source i własne narzędzie – czyli pełne portfolio osoby gotowej do pracy jako Gen-AI Engineer na poziomie mid/regular.

Miesiąc XII — Capstone & Public Launch

W ostatnim miesiącu ścieżki kariery wszystko spinamy w spójną całość. Przeniesiemy najlepsze elementy z poprzednich etapów do jednej, kompletnej aplikacji: z modelem, interfejsem, testami, monitoringiem i wdrożeniem. To będzie Twój projekt końcowy – coś, co możesz pokazać rekruterowi, klientowi lub wrzucić publicznie jako dowód swoich umiejętności.

1. Architektura End-to-End — 2 h

- **Co robisz:** rysujesz diagram (Draw.io / Excalidraw) z przepływem: ETL → Vector DB → LLM API → Front → Monitoring.
- **Cel:** plik capstone_architecture.png w docs/.

2. Integracja modułów (mono-repo hardening) — 6 h

- **Co robisz:** łączysz wszystkie komponenty (backend/, frontend/, infra/) i tworzysz docker-compose.prod.yml.
- **Cel:** make up-prod startuje komplet usług lokalnie w < 60 s.

3. E2E + Security tests — 6 h

- **Co robisz:** playwright: rejestracja → czat → płatność demo; zapscan Docker: brak krytycznych CVE.
- **Cel:** raport e2e_report.html + security_scan.html w reports/.

4. Deploy do chmury + CDN — 6 h

- **Co robisz:** wybierasz jedną z platform: GCP Cloud Run, AWS ECS Fargate lub Azure Container Apps; CloudFront / Cloudflare CDN dla warstwy statycznej.
- **Cel:** publiczny URL https://app.genai-playground.xyz (HTTPS + HTTP/2).

5. Publiczne demo & blog-post launch — 4 h

- **Co robisz:** nagrywasz 90-sek filmik demo, publikujesz wpis „Capstone release” (dev.to / blog własny).
- **Cel:** min. 100 odwiedzin w 48 h; link w README.

6. Feedback & Analytics — 4 h

- **Co robisz:** integrujesz PostHog lub Google Analytics, zbierasz metryki (DAU, tok/s, bounce).
- **Cel:** dashboard analytics_screenshot.png + plan iteracji w journal/.

7. Retrospektywa & Celebration — 4 h

- **Co robisz:** journal/miesiac-12.md — podsumowanie roku, lista „Next 90 days", zdjęcie.
- **Cel:** commit „M-XII: capstone launch".

Jeśli masz ekstra wolny weekend (8h)

Open-source kit „genai-starter-kit" — 8 h

- **Co robisz:** tworzysz osobne repozytorium (genai-starter-kit) z gotowym szablonem aplikacji Gen-AI do dalszego użycia:
  – backend (FastAPI) z przykładowym endpointem /chat i wsparciem dla LLM (OpenAI lub lokalny),
  – frontend (Next.js lub Streamlit) z formularzem promptowania,
  – Dockerfile + docker-compose.yml dla szybkiego uruchomienia.
  Dodajesz dokumentację:
  – instrukcja 1-click deployment (make up, npm install && npm run dev),
  – struktura katalogów, jak podmienić model, jak wdrożyć na Hugging Face / Render.
  Licencja MIT, czytelny README.md, możliwość klonowania i szybkiego startu.
- **Materiały:**
  – Open-source startery: create-t3-app, llm-stack-template,
  – Blog: How to open-source a minimal AI project,
  – Tools: cookiecutter, degit, playwright (do testów)
- **Cel:**
  – repo genai-starter-kit/ z gotowym kodem,
  – README.md z 1-click instrukcją,

– link do repo w genai-playground jako zalecany punkt startowy dla nowych projektów.

Jeśli masz 2 ekstra wolne weekendy (16h)

LLM-powered Auto-Evaluator (RAG + LLM) — 16 h

- **Co robisz:** Tworzymy narzędzie, które automatycznie ocenia jakość techniczną repozytorium na GitHubie z pomocą RAG i LLM.
  – Pipeline:
  –– Użytkownik podaje URL repo lub ścieżkę lokalną,
  –– Skrypt klonuje repo, chunkuje kod (.py, .md, Dockerfile, itp.),
  –– Dane trafiają do wektorowej bazy (Chroma, Qdrant, FAISS),
  –– Użytkownik wybiera typ ewaluacji (techniczna, dokumentacja, architektura, testy),
  –– LLM generuje raport oceny (evaluation.md) z rekomendacjami.
  – Interfejs: CLI (python auto_eval.py --repo ...) lub notebook z GUI (LLMEval.ipynb).
  – Przetestuj na trzech repozytoriach (w tym genai-playground), zapisz wyniki.
- **Materiały:**
  – RAG workflow (np. LangChain, LlamaIndex)
  – transformers, OpenAI, ollama (do inferencji LLM)
  – PyGitHub, os.walk, markdown2
  – Artykuły:
  –– LLM for code review automation
  –– Building LLM-based code analysis tools with embeddings
  –– Opcjonalnie: matplotlib, seaborn do radar chartów
- **Cel:** folder llm_auto_eval/ zawiera:
  – auto_eval.py lub LLMEval.ipynb,
  – config.yaml z promptami i typami ewaluacji,
  – folder results/ z 3 raportami (repo1.md, repo2.md, repo3.md),
  – (opcjonalnie) radar.png z wizualizacją porównawczą.
  Całość gotowa do uruchomienia jako open-source tool lub w CI projektu.

Kamień milowy XII. miesiąca

Ostatni miesiąc to taka checklista, czy rzeczywiście potrafimy to wszystko, co myślimy, że potrafimy...? I zdaje się, że ewidentnie tak! Połączyliśmy backend,

frontend, model, monitoring i zabezpieczenia – i opublikowaliśmy gotowe demo. Jesteśmy gotowi.

Repozytorium *genai-playground* zawiera:

- capstone_architecture.png + docker-compose.prod.yml,
- działający publiczny endpoint (np. Hugging Face Space / Cloud Run) + link w README.md,
- reports/e2e_report.html, reports/security_scan.html (testy end-to-end i bezpieczeństwa),
- demo wideo (90–120 s) + blog-post z linkiem i statystykami,
- screenshot analytics_screenshot.png z narzędzia do analityki (np. PostHog, Plausible),
- journal/miesiac-12.md z retrospektywą i planem dalszego rozwoju,
- (opcjonalnie):
  – folder genai-starter-kit/ – z gotowym szablonem repo do reużycia,
  – folder llm_auto_eval/ – z działającym narzędziem do audytu repo LLM i raportami z testów.

Kamień milowy XII potwierdza, że możesz z dumą nazywać się programist(k)ą AI o specjalizacji Gen-AI. Potrafisz zintegrować kompletny stack technologiczny,, zadbać o jakość i bezpieczeństwo, opublikować projekt i – jeśli trzeba – przeprowadzić jego automatyczny audyt.

Podsumowanie

Czas leci – rok minął jak z bicza strzelił. Jeszcze rok temu myślałeś_aś, że generatywne AI to tylko pornole z celebrytami i fejkowi lekarze wciskający pseudo-leki na Instagramie. I... nie myliłeś_aś się. To też gen-AI. Ale teraz wiesz, że to tylko jedna z jej twarzy – może i najbrzydsza, ale wciąż część większej całości. A całość jest imponująca: piękna, złożona i pełna potencjału do zmieniania rzeczywistości.

Dziś generatywna AI to jedna z najszybciej rosnących dziedzin w obszarze sztucznej inteligencji — od modeli tekstowych, przez generowanie obrazów, po automatyzację pracy biurowej. A Ty w ciągu ostatnich 12 miesięcy stałeś_aś się specjalist(k)ą w tej dziedzinie.

Teraz już wiesz, czym są LoRA, RAG, Guardrails, jesteś za pan brat z bibliotekami Hugging Face. Potrafisz przygotować dane, trenować i ewaluować modele, zabezpieczyć je, wystawić w chmurze lub na Raspberry Pi, a nawet podpiąć płatności Stripe, jeśli trzeba. I oczywiście masz na to dowód w postaci gotowego produktu z modelem, API, UI, testami, monitoringiem i wersją demo.
Brawo!

Co dalej?

Choć ścieżka kariery Gen-AI od porozmawiAImy.pl dobiega końca, to tak naprawdę dopiero początek Twojej kariery. Generative AI to nie jednorazowy trend, a raczej zupełnie nowy sposób pracy z danymi, kodem, obrazem i językiem. I nic nie wskazuje na to, by ta tempo zmian na rynku miało zwolnić.

Jeśli mogę dać Ci jedną radę — znajdź swoją niszę. Gen-AI to tak szeroka specjalizacja (pamiętasz? FullStack AI!), że nie masz szans być ekspertem od wszystkiego. Wybierz sobie jeden, dwa tematy, które naprawdę Cię interesują i bądź w nich na bieżąco. Agentowe automaty? Własne produkty SaaS? Lokalny fine-tuning LLM-ów? Wybór należy do Ciebie — narzędzia już masz.

Z praktycznych rzeczy: Chwal się! Chodź po mieście i krzycz na cały głos, co udało Ci się osiągnąć. Bo masz powód do dumy! Dodaj link do repozytorium na GitHubie do swojego LinkedIna, zrób demo, wrzuć screena z działającej aplikacji, napisz krótkie case study na blogu albo w poście. Umieść projekt w CV — w

sekcji „Najważniejsze osiągnięcia”. To świetny punkt zaczepienia na rozmowy kwalifikacyjne.

Pamiętaj — to Ty jesteś najlepszym ambasadorem Twoich osiągnięć.