Syllabus for the Course "Artificial Intelligence&Large Linguistic Models: Prompt&Context Engineering"

Volume: 7 ECTS credits (252 hours)
Semesters: 1–2

Abstract of the course

1. General characteristics of the discipline

The course is a mandatory part of the general scientific module of the Master's program «AI-Augmented Digital Systems Engineering» in the field of study 09.04.02 «Information Systems and Technologies».

The course is worth 6 credit units (216 academic hours). The assessment structure includes an exam and a term paper/project. The course is delivered in Semesters 1–2 and provides a foundation for subsequent courses in AI engineering, machine learning, and digital systems.

2. The Place of the Discipline in the Program Structure

The course provides the methodological and technological foundation for applying artificial intelligence in digital systems engineering.

It aligns with the introductory and advanced stages of mastering AI technologies (Levels 1–2 of the program's learning model) and facilitates the transition from theoretical understanding of model operation principles to their practical engineering application in real-world systems.

Prerequisites:

- Algorithms and Data Structures;
- Programming Fundamentals;
- Programming Languages.

Courses that build on this discipline:

- Machine Learning and Deep Learning;
- Natural Language Processing;
- Computer Vision;
- Edge AI Technologies;
- Verification and Validation of AI Systems;

- Project Work.

3. Objectives of mastering the discipline

The objectives of mastering the discipline are:

- development of a systemic understanding of the operating principles of modern artificial intelligence models, including large linguistic models (LLM);
- development of skills in engineering application of AI tools in digital systems;
- mastering the methods of prompt engineering and context engineering as tools for managing the behavior of intelligent models;
- preparation for the integration of AI assistants into the development and operation of digital systems.

4. Objectives of the discipline

Upon completion of the course, students will be able to:

- develop a comprehensive understanding of the operating principles of modern artificial intelligence models, including Large Language Models (LLMs);
- acquire practical engineering skills in applying AI tools within digital systems;
- gain proficiency in prompt engineering and context design for guiding AI model responses;
- prepare to integrate AI assistants into the development and operation of digital systems.

5. Planned learning outcomes

Upon successful completion of the course, students will be able to:

- understand the architectural principles of building modern AI systems and Large Language Models (LLMs);
- craft effective, structured, and targeted prompts for solving engineering problems;
- apply context engineering techniques to guide and shape the responses of AI models;
- integrate LLMs into the design, development, deployment, and analysis of digital systems;
- recognize and evaluate the limitations, potential biases, errors, and ethical risks associated with AI technologies;
- critically evaluate and justify the selection of methods and strategies for applying AI in engineering scenarios.

6. Brief Summary of the Discipline

1. Fundamentals of Artificial Intelligence: evolution of approaches, system architecture, and modern trends.
2. Architecture of Large Language Models (LLMs): transformer-based architectures, pre-training, fine-tuning, and inference optimization.
3. Prompt Engineering: principles of query formulation, strategies for managing model outputs, and the use of templates and scripts.
4. Context Engineering: memory management techniques, the role of system instructions, prompt chaining, and retrieval-augmented generation (RAG) approaches.
5. Engineering Scenarios for LLM Application: code generation, system architecture analysis, technical documentation automation, and software testing support.
6. Limitations and Risks: model hallucinations, factual inaccuracies, bias mitigation, safety protocols, and data privacy considerations.
7. Integrating AI into Digital Engineering Processes: methodologies for embedding AI assistants into development workflows and operational systems.
8. Hands-On Learning Components: real-world case studies and practical mini-projects to reinforce key concepts.

7. Educational Technologies

The following teaching and learning methods are employed as part of the course:

- Lectures featuring in-depth analysis of architectural solutions;
- Hands-on workshops on designing effective prompts and interaction scenarios;
- Lab sessions using state-of-the-art Large Language Model (LLM) platforms;
- Mini-projects focused on integrating AI assistants into real-world engineering tasks;
- Case study analysis of practical applications in AI engineering.

8. Methodological Framework of the Course

The course is structured as a dual-track, helix-like model: each learning cycle integrates a deep understanding of the theoretical nature of AI tools with the practical ability to apply and build with them. Theory and practice are interwoven, advancing in parallel and mutually reinforcing and explaining one another.

The core principle of the course is to distinguish between what is technically correct and what merely appears plausible. A student who internalizes this principle becomes a competent AI engineer; one who fails to do so risks becoming a potentially dangerous user of technology.

A capstone end-to-end project begins in Week 4 of the first semester and culminates in a final defense in Week 16 of the second semester. By the end of the program, each student presents a fully working artificial intelligence system. This system includes:

- a retrieval-augmented generation (RAG) knowledge base;
- an AI agent layer (capable of autonomous decision-making and task execution);
- seamless integration into a real-world or simulated business workflow.

Summary table of blocks and results

Block	Weeks	Key Artifact	Learning Outcome / Meaning
1. The Nature of the Instrument	Weeks 1–4, Sem. 1	Prompt specification for a project task	Developing critical thinking: building immunity to uncritical trust in AI outputs
2. Prompt Engineering	Weeks 5–8, Sem. 1	Curated library of structured prompts with behavioral guardrails	Mastering model behavior management through controlled prompting
3. Context and API Integration	Weeks 9–12, Sem. 1	Deployed web-based chat service with API integration	Transition from scripted responses to integrated, API-connected systems
4. Retrieval-Augmented Generation (RAG)	Weeks 13–16, Sem. 1	RAG system with quantified performance metrics	Reducing hallucinations through knowledge-grounded generation and quality measurement
5. Agent Systems	Weeks 1–4, Sem. 2	Tool-integrated agent system with autonomous capabilities	Enabling autonomous actions via agents that leverage external tools and APIs

6. Quality and Safety Assurance	Weeks 5–8, Sem. 2	Comprehensive test suite, performance metrics, and threat model	Achieving production maturity: ensuring reliability, safety, and robustness
7. Advanced Architectures and Observability	Weeks 9–12, Sem. 2	Enhanced AI system with monitoring and observability features	Developing the ability to iterate, debug, and improve complex systems
8. Capstone Project	Weeks 13–16, Sem. 2	Fully documented and tested AI system (portfolio-ready)	Demonstrating comprehensive mastery: a professional portfolio deliverable, not just a course requirement

Glossary of Terms

Token — the smallest unit of text processed by the model.

Vector representation (embedding)— a numerical representation of the meaning of a text in a multidimensional space.

Fine-tuning — the adaptation of a pre-trained model to specialized data.

Pipeline — a sequence of data processing stages.

Logging — the recording of system events for subsequent analysis.

Tracing — tracking the path of a request through system components.

Deployment — the process of launching a system into a production environment.

Fragment / chunking — a unit of text when indexed for searching.

Regression testing — checking that a change has not broken the previous behavior.

Hallucination — a confident but factually incorrect response of the model.

Observability — the ability to understand the internal state of a system based on its external signals.

Decision tree — a structured diagram of choice between alternatives.

Threat modeling — a systematic analysis of attack vectors on a system.

Zero-shot — a mode in which the model is not given a single solution example.

Few-shot — a mode in which the model is given several sample solutions.

Chain-of-thought reasoning — a prompting strategy in which the model explicitly infers the steps of reasoning.

Cross-validation of answers (self-consistency) — generating several answers and choosing the most consistent one.

Structured output (JSON mode) — receiving the model's response in a specified machine-readable format.

Function calling — a mechanism that allows a model to request the execution of external functions.

Tool calling — an extension of function calling: the model selects and calls external tools.

Retrieval-Augmented Generation (RAG) — an architecture in which the model's response is supplemented by retrieved documents.

Reranking — the process of re-sorting the documents found by relevance.

Human-in-the-Loop (HILO) — an architectural pattern that requires human participation at critical points.

Guardrails — instructions and mechanisms that define the boundaries of acceptable model behavior.

Model sycophancy — the tendency of a model to agree with the user rather than provide the correct answer.

Overconfidence — the confident tone of a model when the result is unreliable or erroneous.

Prompt injection — an attack in which user input overrides system instructions.

Context window— the maximum volume of text that the model processes in one request.

Rate limiting— an API mechanism that limits the number of requests per unit of time.

Baseline — initial quality indicators against which improvement is assessed.

Feedback loop— a mechanism for using results to improve subsequent actions.

Language model-based quality assessment (LLM-as-a-judge) — using a language model to automatically assess the quality of answers.

LoRA / QLoRA (Low-Rank Adaptation) — methods of low-rank adaptation of a model during additional training with minimal computational costs.

RLHF (Reinforcement Learning from Human Feedback) — reinforcement learning based on human feedback.

BM25 — a full-text search algorithm that takes into account term frequency and document length.

HyDE (Hypothetical Document Embeddings) — a search method that uses the generation of a hypothetical response document and its indexing.

RAGAS — a framework for automatic system quality assessment with supplements from a knowledge base.

FastAPI — a framework for building web services in Python.

Curriculum schedule for the course

Key:

- **Lecture (Lect.)** — theoretical material delivery.
- **Practical / Lab.** — hands-on activities, simulations and discussions.
- **Self-directed work (SDS)** — independent study on project tasks and analysis.

LLM use — applying Large Language Models for analysis, generation and decision support (integrated into independent study tasks).

Calendar-Thematic Plan (1 Semester)

Week	Content	Lect. (h)	Practice (h)	Lab (h)	SDS (h)
Block 1. The Nature of the Tool (Weeks 1–4)					
1	**What Is a Large Language Model Really?** Topic: The evolution of artificial intelligence — from rule-based systems to neural networks and transformers. Core concepts under the hood: tokens, probabilities, and sampling parameters. Why the model doesn't "know" anything — it merely predicts the next token. Demystifying the technology. **Masterclass:** The instructor demonstrates the same task with different values of the variability parameters (temperature) and sampling width (top-p). Students observe how the model's perceived "confidence" varies with these parameters, not with ground truth. Next, a deliberate hallucination is introduced: a request for a non-existent article with a real identifier. Follow-up analysis: why did the model "agree" to generate this false information? **Lab:** Students reproduce a hallucination using a given template. They document the specific conditions under which the hallucination occurs and formulate a hypothesis: under what parameter settings is the risk of hallucination higher? **SDS (Self-Directed Study):** A reflective note — one real-life example of uncritical trust in a language model from the student's own experience. The note should describe: what happened; which information was not verified before trusting the model's output. **Outcome of this stage:** The student understands that a large language model is a probabilistic predictor, not a knowledge base. This insight forms the foundation for developing a critical approach to using LLMs. Without this critical mindset, the entire course would merely teach uncritical use of AI tools	2	2	0	4

Week	Content	Lect. (h)	Practice (h)	Lab (h)	SDS (h)
2	**Transformer Architecture: What an Engineer Needs to Know** **Topic:** The attention mechanism — explained intuitively, without matrices. Context window as an architectural constraint. Tokenization and its implications for prompting (including numbers, code, and languages with non-Latin alphabets). Solution space map: pretraining → inference → fine-tuning → retrieval-augmented generation (RAG). **Masterclass:** Demonstration of the lost-in-the-middle effect. Students perform a single task where key information is placed at the beginning, middle, and end of the prompt. They measure and compare the quality of the model's responses in each case. This exercise allows students to directly observe the architectural constraint in action. **Lab:** Tokenization experiment. Students measure the token cost of different data formats (JSON, YAML, plain text, table) for the same semantic content. They construct a table with the following structure: "format → tokens → cost". **SDS (Self-Directed Study):** Technology application map. Students identify what types of problems are best solved by: prompting; fine-tuning; retrieval-augmented generation (RAG); agent-based systems. For each category, they justify three concrete examples from their subject area, explaining why the chosen approach is optimal. **Outcome of this stage:** The student understands architectural constraints as engineering parameters that can be measured, controlled, and optimized. They no longer perceive the context window as a simple setting, but as a fundamental architectural feature with measurable trade-offs in cost, quality, and latency.	2	2	0	4

3	**Model Landscape: Choice as an Engineering Decision** **Topic:** An overview of modern large language models: GPT-4, Claude, Gemini, Llama, Mistral, GigaChat. Optimization techniques — distillation, quantization, weight pruning: understanding the trade-offs (what we lose, what we gain). Cloud vs. on-premises deployment considerations: economics, latency, privacy. Installing and running Ollama for local model deployment. **Masterclass:** Comparative benchmark. Students run a single task — engineering code analysis — on three models of different sizes (small, medium, large). Measurements include: − response quality (accuracy and completeness); − inference speed (latency in seconds); − computational cost (tokens per query, API pricing). Students analyze the results and draw the conclusion that the "best" model depends on the specific task requirements and operational constraints. **Lab:** Model comparison exercise. Students: Run Llama and Mistral locally via Ollama on a specific task from their project. Compare the local results with those obtained from cloud APIs (e.g., OpenAI, Anthropic) on the same task. Fill out a decision matrix with the following criteria: − performance (quality score); − latency (response time); − cost per query; − data privacy requirements; − infrastructure setup complexity. Justify their choice with quantitative data and write a brief recommendation: when to use a local model vs. a cloud API. **SDS (Self-Directed Study):** Cost analysis project. Students take a real-world scenario (100 queries per day, average context of 2 000 tokens) and calculate the monthly cost for three different APIs (e.g., GPT-4, Claude 3, Gemini Pro). They: − gather current pricing data from official sources; − compute total monthly costs under the given scenario; − estimate the break-even point: at what query volume (per month) does a local model deployment become more cost-effective than cloud APIs? − present findings in a short report with a clear conclusion.	2	2	0	4

Week	Content	Lect. (h)	Practice (h)	Lab (h)	SDS (h)
	Outcome of this stage: The student is able to select a language model as an engineering solution with justified tradeoffs — considering performance, cost, latency, and privacy — rather than simply choosing the most well-known one. They understand how to make data-driven decisions based on quantitative metrics and business requirements.				

	Prompt as an Engineering Specification — Introduction. Starting an End-to-End Project **Topic:** A prompt is not a simple request, but a precise specification of desired behavior. Prompt structure components: − role (defining the model's persona); − task (clear action to be performed); − context (relevant background information); − format (required output structure); − constraints (limitations and rules). Output management approaches: − zero-shot prompting; − few-shot prompting (with examples); − chain-of-thought prompting (step-by-step reasoning). Introduction of the end-to-end project — a capstone initiative that will develop over 28 weeks. **Masterclass:** Live Prompt Refactoring. The instructor: Selects a poorly constructed prompt from a real-world case study. Demonstrates incremental improvements through four iterative refactoring steps. Provides commentary on each design decision, explaining the engineering rationale. Students observe the systematic engineering process behind effective prompting, rather than viewing it as "AI magic". **Lab:** End-to-end project launch. Students: Select a subject area relevant to their professional or academic interests. Formulate a specific problem statement for their future AI system. Write the first system prompt using the structure components covered in the topic. Submit the prompt to a language model and analyze the initial output. Revise the prompt based on the results and feedback. Document all iterations, including: − version number; − changes made; − observed improvements in output quality; − remaining issues to address. **SDS** (Self-Directed Study): Reflective note on the first month of study. Students write a structured reflection addressing: − What has changed in their understanding of language patterns and model behavior compared to week 1?	2	2	2	4

Week	Content	Lect. (h)	Practice (h)	Lab (h)	SDS (h)
	− Which concepts now seem obvious or intuitive that were unclear initially? − What aspects of LLM behavior most surprised them? − How has their approach to interacting with language models evolved? **Outcome of this stage:** The student: − has formulated a clear objective for their end-to-end project; − has written and iteratively improved the first working system prompt; − understands that effective prompting is an iterative engineering process requiring testing and refinement; − possesses a documented development history of their initial prompt iterations; − is prepared to build upon this foundation over the next 28 weeks of the course.				
Block 2. Prompt Engineering as a Discipline (Weeks 5–8) **Why: A prompt is a model management interface. An engineer who can clearly formulate prompts receives a fundamentally different tool than a user who simply writes them outright.**					
5	**Model Output Control Strategies** **Topic.** Comparison of approaches: without examples vs. with examples - when each works better. Step-by-step reasoning (chain - of - thought): why explicit reasoning improves accuracy and when it doesn't work. Cross-validation of answers (self - consistency). Structured inference (structured) output , JSON mode): getting a machine-readable response. Calling functions (function calling) as a prompt discipline. **Masterclass.** One logical inference problem: solved without examples (error), with examples (better), and with step-by-step reasoning (correct). Analysis: what exactly changes in the model's behavior and why. **Lab.** Students apply three strategies to a problem from their project. They document which strategy yielded the best results and why. Mandatory verification: the student independently checks the correctness of each answer, not relying on the confident tone of the model. **SDS.** Prepare a library of prompts for your project: at least five templates for different types of tasks.	2	2	0	4

Week	Content	Lect. (h)	Practice (h)	Lab (h)	SDS (h)
	Outcome of this stage: The student consciously selects a prompting strategy for the task at hand, rather than applying one technique to everything.				
6	**System Instructions and Behavior Control** **Topic:** System prompt as a "constitution" of the model. Delineation of roles: system - user - assistant. Designing a persona engineering). Guardrails in the prompt . Implementation in the prompt injection) - attack and defense. **Masterclass.** Demonstration of a prompt injection attack: students see how user input can override system instructions. Then, defensive techniques are introduced: instruction prioritization, data and instruction separation, and input validation. **Lab. Students write a system prompt for their project with explicit behavior constraints. Then they independently attempt to "break" it using** adversarial input. inputs). Vulnerabilities are documented and fixed. **SDS.** Update the end-to-end project: add a system prompt with protection. Reflective note: which attacks were successfully reproduced, how were they protected. **Outcome of this stage:** The student understands that the system prompt is not a "setting," but a security contract that must be verified.	2	2	2	4
7	**Dealing with Uncertainty and Model Errors** **Topic. Typology of errors in language models: hallucinations** of facts, hallucinations of reasoning, overconfidence , sycophancy . Calibration - when the model "knows that it doesn't know." Verification techniques: querying sources, step - back questioning prompting), the role of the opponent (devil 's advocate). **Masterclass.** A live analysis of acquiescence: the teacher poses a false statement with a confident tone, and the model agrees. Then, there are overcoming techniques: how to get the model to disagree rather than agree. **Lab.** Students stress-test their proposal : they present false assumptions, incomplete data, and contradictory requirements. They document where the model correctly expresses uncertainty, and where it is falsely confident. **SDS.** Write a verification protocol for your project: how will the system user know when a response can be trusted and when it can't? **Outcome of this stage:** The student is able to diagnose model unreliability and design systems with explicit trust boundaries.	2	2	0	4

Week	Content	Lect. (h)	Practice (h)	Lab (h)	SDS (h)
8	**Prompt Engineering: Final Lab** **Topic.** Integration lesson. No new theory is introduced, only reinforcement through a complex practical task. **Masterclass.** Analysis of a real industrial production project (anonymized case study). Students analyze its structure, identify weaknesses, and suggest improvements. **Lab.** Students present the current state of their project's prompt library. Peer review in pairs using a checklist: structure, behavior constraints, verification, error handling. The language model is used as a secondary reviewer, but not as the primary one. **SDS.** The final version of the project's prompt library with documentation of changes following the review. **Outcome of this stage:** The student has a working, peer-reviewed library of prompts—the first significant artifact of the end-to-end project is complete.	0	2	2	6
Block 3. Context Engineering and Working with the API (Weeks 9–12) **Why: A prompt without context management is a script. Context engineering transforms individual requests into a system with memory, state, and manageable behavior.**					
9	**Context as a Managed Resource** **Topic:** Context window - physics and economics. Memory management strategies: sliding window, summation , selective storage Retention). The lost - in- the - middle effect — how the position of information affects quality. Context as an architectural solution. **Masterclass.** Experiment: A long dialogue with escalating context—at what point does the model "forget" the beginning? How do different context compression strategies affect the quality of responses? **Lab.** Students implement two context management options for their project (sliding window vs. summation), measure quality and cost, and select a strategy with justification. **SDS.** Update the project architecture: how will memory be managed in the final system? **Outcome of this stage:** The student designs the context as an architectural component, rather than leaving it to the model.	2	2	0	4
10	**Integration with the API: From Prompt to System** **Topic:** OpenAI API , OpenRouter , GigaChat — request structure, message roles, parameters. Streaming . Request rate limiting . limiting) and retry logic (retry Logic). Error handling. Response caching: when and why. Cost optimization.	2	2	2	4

Week	Content	Lect. (h)	Practice (h)	Lab (h)	SDS (h)
	Masterclass. Live building of a client API from scratch: from a simple request to exponential backoff retry logic backoff), caching. Students see how production code differs from the training example. **Lab.** Students implement API integration for their project: provider selection with justification, retry logic, basic caching. They measure the cost of 100 test requests and document their results. **SDS.** Calculate the projected monthly cost of your system under three load scenarios. At what load threshold does the system become economically unfeasible? **Outcome of this stage:** The student is able to build a reliable API client that handles real production problems, not just mockups.				
11	**FastAPI + Language Model: First Application** **Topic:** Architectural patterns for AI-based applications. FastAPI as a wrapper for a language model service. Asynchronous request processing. Separation of prompt logic from application logic. Configuration via environment variables. variables). **Masterclass.** Building a minimal AI service using FastAPI : an endpoint receives a request, generates a prompt, calls a language model, and returns a response. Then comes refactoring: extracting prompt templates, asynchronicity, and basic logging. **Lab.** Students wrap their prompt logic in the FastAPI service. They test it using standard tools. They document the API's software contract. contract). **SDS.** Write documentation for your service: what it does, how to launch it, what parameters it accepts, how it handles errors. **Outcome of this stage:** The end-to-end project becomes a executable service for the first time, not just a set of prompts. The student has gone from "write a request in ChatGPT " to "implement a programming interface."	2	2	0	4
12	**Chatbots and wealth management** **Topic:** Conversational AI Architecture: Stateless vs. Stateful vs Stateful). Dialogue history management. Processing of different message types. Integration with Telegram and corporate platforms. Personalization through user context. **Masterclass.** Building a bot with memory: from stateless mode (each request is independent) to stateful mode (the bot remembers the session context). Demonstration of degradation due to improper memory management.	2	2	2	4

Week	Content	Lect. (h)	Practice (h)	Lab (h)	SDS (h)
	Lab. Students add a conversational interface to their project (Telegram or web chat). They implement session context persistence. They test edge cases: very long conversations, topic changes, and conflicting requests. **SDS.** User testing: Ask three people to talk to the bot. Record three unexpected behavior scenarios. Analyze the causes. **Outcome of this stage:** the end-to-end project now has a user interface. The student sees their system through the eyes of a real user for the first time—and that's always unexpected.				
Block 4. Semantic search and generation with addition from the knowledge base (weeks 13–16) Why: A language model without external knowledge is a closed system. Generation with supplementation from the knowledge base opens the model to current, specialized, and verified data. This is key. architecture majority production AI systems .					
13	**Vector representations and semantic search** **Topic:** Embedding concept : meaning as a point in a multidimensional space. Cosine similarity Similarity). Why semantic search outperforms full-text search for fuzzy queries. Vector databases: Qdrant , pgvector , Pinecone — comparison. **Master class.** Visualization: phrases with similar meanings are clustered in a vector representation space. Live experiment: the same query through keyword search and semantic search – different results, different errors. **Lab.** Students implement semantic search on a small corpus of documents in their subject area. They compare precision and recall with keyword searching and document the cases in which each approach is superior. **SDS.** Prepare a set of documents for your project: what exactly will the system know? How will it be updated? **Outcome of this stage:** The student understands vector representations not as "magic," but as a metric space with its own geometry—and knows how to use it.	2	2	0	4
14	**System architecture with add-on: indexing and searching** **Topic:** Retrieval - Augmented Generation Pipeline (RAG) Generation : chunking → vectorization → indexing → extraction → augmentation → generation. Chunking strategies: fixed size, semantic, hierarchical. Impact of chunk size on quality . Reranking . **Masterclass.** A live demonstration of the impact of a breakdown strategy on answer quality: one document, three strategies—students see different answers to a single question and analyze the reasons.	2	2	0	4

Week	Content	Lect. (h)	Practice (h)	Lab (h)	SDS (h)
	Lab. Students implement a basic generation pipeline with an add-on for their project. They experiment with fragment sizes. They measure the quality of answers to five test questions. They find the optimum. **SDS Study.** Write 20 test questions for your system: 10 should be answered correctly according to the documentation, 10 should deliberately go beyond the system's knowledge. How does the system behave in both cases? **Outcome of this stage::** the student has a working system with a knowledge base supplement for their project. This is a qualitative leap: the system responds based on real documents, not the "fantasies" of the model.				
15	**System Quality with Add-On: Measurement and Improvement** **Topic.** Quality metrics: reliability , relevance of the answer relevance), context accuracy (context precision), completeness of context (context Recall). Hallucinations in systems with knowledge base augmentation are a specific issue. Typical errors include poor extraction, loss of context during generation, and source conflicts. **Masterclass.** Analyzing a real-life failure: the system reliably responds based on an irrelevant fragment. Diagnostics: How to detect? What to fix? **Lab.** Students systematically test their system using 20 questions from the previous week's SDSbook. They fill out a table: question → found fragment → answer → accuracy assessment. They identify error patterns. **SDS.** Prepare a report on the system's quality, including details on what works, what doesn't, and what improvements are planned for the second semester. **Outcome of this stage:** The student can not only build a system with an add-on, but also measure its quality and diagnose errors. This is the difference . between demo and production system.	0	2	2	8
16	**Final laboratory of the first semester** **Subject:** No new theory. Integration of everything that has been constructed. **Masterclass.** Analyzing common mistakes at the end of the semester, using examples from student projects (with the authors' permission). What was missing? What turned out to be more difficult than expected? **Lab.** Project pre-defense in a "3-minute demo + 5-minute questions" format. Focus: does the system work, and can the quality of its responses be measured?	0	2	0	6

Week	Content	Lect. (h)	Practice (h)	Lab (h)	SDS (h)
	Exam section (SDS + consultation). Individual interview: the student explains the architecture of their system, answers questions about the nature of each component, and demonstrates an understanding of its limitations. **Outcome of this stage:.** By the end of the first semester, the student has: a working generation service with a knowledge base add-on and a dialog interface, a library of prompts, measured quality, and an understanding of the limitations. This isn't just a mockup—it's a prototype that can be shown to an employer.				

Calendar and thematic plan for SEMESTER 2. "Complexify the system and put it into production"

Ned.	Topic	Lec.	Practical.	Lab.	SDS
1	**Agent concept and tool invocation** **Topic:** Agent as a language model + tools + cycle. Calling tools (tool) calling): architecture and protocol. Pattern comparison: ReAct , plan-and-execute, reflexive agent. When an agent is needed and when it's redundant. **Masterclass.** Live construction of a minimal agent: language model + document search tool. Students see the Observe → Think → Act cycle in real time—and its failures. **Lab.** Students add their first tool to their project: knowledge base search via tool invocation. They test: when does the agent decide to use the tool, and when does it respond from "memory?" **SDS.** Design a toolkit for your project: what exactly should the agent be able to do? What external systems should it call? **Outcome of this stage:** The student understands the agent as an architectural pattern, not as a marketing term.	2	2	0	4
2	**Agent reliability and error management** **Topic:** Typical agent failures: infinite loops, incorrect tool usage misuse), calling a non-existent instrument (hallucinated tool call), error accumulation. Constraint strategies: maximum number of iterations, confidence thresholds, human-in- the - loop control . Logging and tracing of agent behavior.	2	2	0	4

	Masterclass. Demonstration of a "broken" agent: a deliberately created infinite loop scenario and its diagnostics using logs. How to add protective mechanisms. **Lab.** Students deliberately break their agent, finding failure conditions. Then they add protective mechanisms. Mandatory: logging every step of the agent for subsequent diagnostics. **SDS.** Write a troubleshooting guide (runbook) for your agent: what to do if the agent gets stuck, gives an incorrect response, or calls a non-existent tool? **Outcome of this stage::** The student builds agents with explicit reliability boundaries—they don't "trust" the agent, but rather design its behavior in boundary cases.				
3	**Multi-agent systems** **Theme .** Orchestration agents : pattern supervisor (supervisor pattern), pipeline , pattern Debate patterns . Division of responsibility between agents. Communication between agents. When multi-agency is justified, and when it complicates things unnecessarily. **Masterclass.** Building a two-agent system: a research agent + an editor agent. Students will see how separating responsibilities improves the quality of the result—and the cost of coordination in tokens. **Lab.** Students decompose their project's task into two or three agents with different roles. They implement minimal orchestration . They measure: has the quality improved? How much does an additional agent cost? **SDS.** Update the project's architectural diagram: what does the system look like now? What agents, what tools, what data flows? **Outcome of this stage:** The student is able to design multi-agent systems with rationale—not just "add agents," but with an understanding of the tradeoffs.	2	2	0	4
4	**Integration of agents with external systems** **Topic:** Agent as an orchestrator of external APIs. Integration with databases, web search, and enterprise systems. Security: what the agent is and is not allowed to do. Agent activity log (audit) log). **Masterclass.** An agent with access to a real open API. Demonstration: how the agent decides which tool to call, how it handles external service errors. **Lab.** Students integrate at least one external data source into their agent. They implement a basic log: a record of each external call with a timestamp, parameters, and result.	2	2	2	4

	SDS. Conduct a security analysis of your system: what can an agent do without the user's explicit permission? What are the limits of its authority? **Outcome of this stage:** The end-to-end project has evolved into an agent-based system with external integrations. Student can design powers agent How architectural solution .				
Block 6. Quality, Validation, and Manufacturing Maturity (Weeks 5–8) **Why: A system that can't be measured is a system that can't be improved. Production readiness isn't just "it works on my laptop"; it's working under load, measurably, and with manageable risks.**					
5	**Week 5: AI-Based Systems Quality Metrics** **Topic.** What does a "good answer" mean? Automatic metrics: RAGAS , G - Eval , language model-based quality assessment (LLM - as - judge). Manual evaluation: markup guidelines, inter - rater agreement agreement). When each approach works. Evaluation sets (evaluation datasets) - how to create and maintain. **Masterclass.** Live comparison: one system assessed automatically and manually. Where do the metrics match the human assessment, and where do they diverge? Why? **Lab.** Students create a set of questions to evaluate their project: 30 questions with standard answers. They run an automated assessment using a language model for quality control. They compare the results with their own assessment. **CRS.** Establish a baseline for your system's quality: current performance on selected metrics. This is the starting point for all subsequent improvements. **Outcome of this stage:** The student is able to measure the quality of their system—not "it seems to work fine," but "precision 0.73, recall 0.81, accuracy 0.89."	2	2	0	4
6	**Validation and testing of AI systems** **Topic: Specifics of testing non-deterministic systems.** Unit tests tests) for prompts. Integration tests for pipelines. Regression testing A / B testing : how to make sure an improvement doesn't break something else. testing) prompts. **Masterclass.** Building a test infrastructure for an AI service: from "run and see" to automated tests with quality checks (assertions). **Lab.** Students write a minimal set of tests for their project: 10 unit tests for prompts and 5 integration tests for the pipeline. They configure automatic execution. **SDS.** Conduct regression testing: make one change to the prompt, run tests, and document the impact on quality across all metrics.	2	2	2	4

	Outcome of this stage: The student builds an AI system with an engineering culture of testing—they're not afraid to make changes because they know how to measure their consequences.				
7	**Risk Management and Security** **Topic:** Classification of AI system risks: technical, operational, ethical. Implementation in prompt injection in production. Data leakage through a language model. Personal data in context. Content moderation moderation). Compliance and audit (compliance). **Masterclass.** Threat modeling Modeling for an AI system: Students and their instructor walk through the architecture of a typical agent with a knowledge base and identify attack vectors. A countermeasure is developed for each. **Lab.** Students conduct threat modeling of their project using a template. They identify at least five risks, each with an implemented or planned countermeasure. They document these in a security log. **SDS.** Write a policy for using your system: what the user can and cannot do, and how the system responds to violations. **Outcome of this stage:** The student designs security as an architectural component, rather than adding it on as an afterthought.	2	2	0	4
8	**Language model in development processes and build pipelines** **Topic: Language Model as a Developer's Tool:** Code Review review), test generation, documentation, architecture analysis. Integration into the continuous integration and delivery pipeline (CI / CD) pipeline). Automation via GitHub Actions . Limitations: When the language model in the pipeline is dangerous. **Masterclass:** Live Integration of a Language Model into GitHub Actions : Automatic code review for each pull request Students see what errors the model finds, what it misses, and what generates false positives. **Lab.** Students add automated language model-based validation to their project's pipeline: pull request description generation, basic code review, and implementation prompt verification. **SDS.** Evaluate: How useful is the language model in your project's pipeline against noise generation? Where is the limit of reasonable use? **Outcome of this stage:** The student is able to integrate a language model into engineering processes with an	0	2	2	6

	understanding of limitations—not "I automate everything," but "I automate where it reduces risks."				
9	**Advanced Search with Add-on: Hybrid Search and Reranking** **Subject:** Hybrid Search search) : the BM 25 full-text search algorithm combined with semantic search. Reranking models . Search through hypothetical documents (HyDE - Hypothetical Document Embeddings : generation of a hypothetical answer and its indexing. Corrective and self-correcting generation with addition (Corrective RAG , Self - RAG). When every complication is justified. **Masterclass.** Comparison: Basic search with augmentation vs. hybrid search vs. hybrid search with reranking on the same dataset. Students see the improvement in quality—and its cost in latency and money. **Lab.** Students improve the system by adding a hybrid search or re-ranking feature to their project . They measure the change using the quality metrics established in week 5. **SDS.** Update the project quality report: What has changed since the search system was improved? **Outcome of this stage:** The student is able to iteratively improve the AI system based on measurements rather than intuition.	2	2	0	4
10	**Multimodal scenarios** **Topic:** Multimodal Language Models (vision) LLM): analysis of documents, diagrams, and interfaces. Audio models . Multimodal pipelines. Specifics of prompting for working with images. Limitations and common pitfalls. **Master class.** Building an intelligent document processing pipeline (document intelligence pipeline): document image → text recognition via a multimodal model → structured data → question answering. Comparison with traditional character recognition (OCR). **Lab.** Students add multimodal capabilities to their project (if applicable) or explore a multimodal scenario for a related problem. They document the limitations. **SDS.** Mapping the applicability of multimodality in its subject area: where does working with images or audio add real value, and where is it redundant? **Outcome of this stage:** The student understands multimodality as an architectural choice with specific tradeoffs.	2	2	2	4
11	**Further training and adaptation of models** **Topic:** When fine - tuning is necessary—and when it isn't. The low-rank adaptation method LoRA and its lightweight version	2	2	0	4

	QLoRA (Low - Rank) Adaptation) — intuition: what we lose, what we gain. Setting up the following of instructions (instruction) tuning) versus reinforcement learning based on human feedback (RLHF - Reinforcement Learning) Learning from Human Feedback). Cost and infrastructure. Alternatives: example-based approach, generation with addition, system prompt—which is cheaper for what data volume? **Masterclass.** Comparison: one problem is solved by prompting , generation with augmentation, and retraining. Students see real tradeoffs—not in theory, but in terms of quality, time, and cost. **Lab.** Students analyze whether their project requires additional training. They build a decision tree . tree): under what conditions is prompting /generation with padding/retraining the optimal choice. **SDS.** Final architectural justification for the project: rationale for all key choices (model, augmented vs. retrained generation, agents vs. pipeline, cloud vs. on-premises deployment). **Outcome of this stage:** The student is able to justify the architectural decisions of their AI system—not just because "it just happened that way," but because "it's optimal given the constraints."				
12	**Observability of AI systems** **Topic:** Monitoring AI in Production: What to Log , How to Set Up Alerts, and How to Debug. Tracing Language Models: Tools like LangSmith and Langfuse . Quality Drift Drift : How to detect system degradation. Feedback loops loops). **Masterclass.** Connecting tracing to an existing application. Students will see every language model call from the inside: prompt, response, latency, cost, and quality score. **Lab.** Students add observability to their project: tracing each language model call, basic metrics, and a dashboard . They simulate quality degradation and detect it using metrics. **SDS.** Write a troubleshooting guide for a production incident: system quality has dropped by 20%. How to detect, diagnose, and fix it? **Outcome of this stage:** The student is able to support an AI system in a production environment—not only building it, but also maintaining it.	2	2	2	4
Block 8. Final Project (Weeks 13–16) **Why: Integrating everything. The student presents not a set of lab assignments, but a system**.					

13	**Architectural audit and final design** **Master class.** The instructor conducts a public audit of two student projects (with the authors' consent), identifies architectural risks, and suggests improvements. The format is a real-life architectural review. **Laboratory.** Mutual architectural audit in pairs using a checklist: components, quality metrics, security, observability, operating costs. **SDS.** Final plan of improvements based on the audit results. **Outcome of this stage:** The student receives an external review of their system and a list of specific improvements.	2	2	0	4
14	**Implementation of final modifications** **Laboratory.** Project work. The instructor acts as a consultant. **SDS.** Writing final documentation: architecture, quality, limitations, operating costs, further development plan (roadmap). **Outcome of this stage:** The system is technically and documentarily ready for defense.	2	2	0	4
15	**Preparing for defense** **Laboratory.** Defense run-through in small groups (3-4 people). Feedback from colleagues. Final edits. **SDS.** Preparation of presentation and demonstration. **Outcome of this stage:** The student is ready to defend not only the system's operation but also each architectural decision.	0	2	2	8