NATURAL LANGUAGE PROCESSING
FINAL PROJECT GUIDE AND CHALLENGE TRACKS

Benyou Wang[*], **Minghao Wu, Shunian Chen, Ke Ji, Sirui Li,**
Kuang Wang, Juhao Liang, Siqi Rao, Aoxiang Qin
The Chinese University of Hong Kong, Shenzhen
wangbenyou@cuhk.edu.cn

ABSTRACT

The final project is a small research study in natural language processing. Choose one question, build a baseline, and test a targeted change under a fixed budget. Your final deliverables are a poster presentation, a research report, and reproducibility materials. Projects may build a system, reproduce and extend a method, or investigate a competition task. This guide covers CSC5051, CSC6052, and MDS5110.

1 SCHEDULE

Course dates below follow the Fall 2026 course website. Course times are in Shenzhen local time (UTC+8). Follow course announcements for submission links and presentation logistics.

Date	Milestone
October 30	Final project released
November 6	LLM Agents and Harness Engineering lecture
November 13	Final Projects and Research Sharing. Team formation and optional proposal due.
December 17, 17:00	Poster submission due
December 18	Final project poster presentations
December 25, 23:59	Final report due

A suggested four-week research window runs from November 13 to December 11. Use week 1 to freeze the task, evaluation split, and baseline. Use week 2 to implement the main change. Use week 3 for matched comparisons and error analysis. Use week 4 to reproduce the key result and prepare the demo or result figure. Reserve the remaining time for the poster and report.

2 TEAMS AND TOPIC SELECTION

You may work alone or in a team. The suggested team size is two or three. If you need a larger team, explain how the workload and individual responsibilities justify it. Each student is assessed individually. Identify your own work if you collaborate with people outside the course.

Choose a project format that fits your question and resources. The appendices give project briefs and evaluation plans.

- **Build and evaluate a system.** Choose a featured prototype from Appendix A, or propose a related NLP application.
- **Reproduce and extend a method or benchmark.** Select a bounded study from Appendix B.
- **Run a competition project.** Select a live contest or an archived benchmark from Appendix C and evaluate one research change against a baseline.

[*] Course instructor.

- **Investigate a focused research question.** Choose a project from Appendices D–G based on what you plan to change and measure.

All project options use the same course schedule, deliverables, and grading criteria.

An optional one-page proposal is useful when the research question, evaluation method, or compute budget needs early feedback. State the question and motivation, closest prior work, data or environment, baseline, proposed change, primary metric, resource budget, and a smaller fallback experiment. Submit it through the channel announced by the teaching team.

3 Experiment Design

Begin with one working baseline and one measurable outcome. Compare your method on the same held-out tasks and report the variables that affect interpretation, such as model version, prompts, training tokens, time limits, hardware, or API spend. Include at least one comparison that isolates the main change, and show both a success and a failure with your explanation.

An API model such as Claude Opus 5.5 can act as a planner, tool caller, or research assistant. Log the model and its role. Where possible, verify results with executable tests, ground truth, or human review. For hosted competitions, use models and tools allowed by the organizer.

4 Deliverables and Assessment

- **Poster and presentation.** State the question, baseline, main result, and one limitation. A live demo or recorded replay can illustrate the result. A sample poster template is available.
- **Final report.** Submit a PDF of at most six pages of main text using the course report template. Cover the motivation, related work, method, evaluation, results, failures, and limitations. Include a contribution statement specifying each member's work.
- **Reproducibility.** Submit code on GitHub with a README, environment and model versions, evaluation inputs or instructions for obtaining them, and the commands to reproduce the main result. In the report, describe material uses of generative AI in coding, data, evaluation, and writing.

The final project contributes 55 percentage points to the course grade. The poster contributes 10 points and the report contributes 45 points. The breakdown below follows the course grading policy.

Component	Criteria and points
Poster (10)	Poster quality (1), oral presentation (4), overall assessment (5)
Report (45)	Technical interest (10), technical soundness (15), writing clarity (15), individual contribution (5)

Other adjustments and the 55-point cap follow the course website. Reports are made public after the poster session unless you notify the teaching team. If you release your work after the course, please acknowledge the course.

Example contribution statement. Alice designed the evaluation and annotated the held-out cases. Bob implemented the baseline and the proposed method. Both ran experiments and wrote the report.

A Featured Projects

Choose a brief below or use the linked research appendices to scope a related question. Each brief specifies a four-week prototype and a comparison suitable for the report. The replication and competition briefs begin in Appendices B and C.

Project group	Research focus
Interactive 3D worlds and characters	Spatial language, agent memory, and human interaction (Appendices E, F)
Generation and iterative repair	Executable feedback, search, and held-out tests (Appendices E, G)
Generative AI for education	Source fidelity, interactive simulation, and transfer (Appendix F)
Training, model internals, and serving	Data, interventions, and systems measurement (Appendices D, G)
Evidence and verification	Claim-level support, provenance, and abstention (Appendices E, G)

For 3D studies, start with one small environment and reuse assets. Hold out layouts, tasks, or requests. Keep scene access identical between methods unless access itself is the variable. A separate checker may use simulator state to score the results. For course projects, supplied social-media links serve as design prompts. The linked posts could not be independently checked during preparation.

A.1 Interactive 3D worlds and characters

A GTA-inspired CUHK-Shenzhen campus with LLM characters. Build a playable slice of the campus with a few recognizable landmarks, two NPCs, and three tasks such as finding a room, asking about an event, and following directions. Compare a scripted NPC, an LLM with a static campus briefing, and the same LLM with state-aware retrieval of approved campus facts. Give both LLM variants the same landmark queries and game commands. On held-out routes and dialogue scenarios, measure factual accuracy, valid actions, quest completion, and response latency. Use week 1 for the map and scripted baseline, week 2 for the dialogue and action interface, week 3 for tests, and week 4 for a playable build and replays. See Generative Agents for a reference architecture for character memory.

A three.js world with AI-generated assets that adapts to its user. Build one browser scene and generate a small library of assets for it, then import the meshes as glTF. Compare a fixed layout with two methods that adapt the same asset library to user requests, one using free-text preferences and one using structured preferences. Give both variants the same preview and undo controls. Measure the fraction of requests satisfied, unintended scene changes, user corrections, asset load failures, and frame time. In week 1 define the asset and preference schema, in week 2 implement the two variants, in week 3 run the same requests through all variants, and in week 4 deliver the playable scene and evaluation traces. See the inspiration post and three.js model-loading guide.

A digital human that dances to audio. Retarget a small library of motion clips to a single rigged avatar. Compare fixed playback, beat-matched retrieval, and retrieval conditioned on both audio and a short style instruction, using held-out tracks. Measure the alignment between movement accents and audio beats, transition jerk or foot sliding, and blinded style preference. Use week 1 to get the rig and baseline working, week 2 to add audio and language conditioning, week 3 to score 8–12 short clips, and week 4 to deliver a browser or recorded demo with matched comparisons. See the inspiration post and AI Choreographer and AIST++.

A 3D assistant that asks which object you mean. In a furnished room with several similar chairs, compare a policy that picks its best guess, one that always asks, and one that asks only when competing objects remain plausible. Use the same model and scene observations for all three policies, then measure correct selections and questions asked on held-out rooms. Deliver a browser demo and a replayable test set with ambiguous and unambiguous commands. See ReferIt3D.

Multiturn language editing for 3D scenes. Compare direct generation of scene edits with a typed action schema and a geometry checker on requests such as *Move the chair beside the desk without blocking the door.* Include follow-up references and corrections, and measure relation satisfaction, unintended edits, and invalid placements. Deliver an editor that reuses existing assets and replays its edit history. See EditRoom.

An embodied agent that chooses where to look. Ask questions whose answers are initially occluded and compare a fixed viewpoint, random exploration, and question-guided camera actions under the same action budget. Measure answer accuracy, observations used, and navigation steps in held-out rooms, using a simulator state checker for scoring. Deliver a replay showing how the agent moves, observes, and answers. See AI2-THOR.

A.2 Generation and iterative repair

A procedural pelican on a bicycle and a hand-built evaluation harness. Adapt Simon Willison's drawing challenge to a browser 3D coding task. Give the model the following prompt and require a single runnable HTML file.

> Create a polished, interactive three.js scene of a pelican riding a bicycle. Construct the pelican, bicycle, and environment with procedural 3D geometry. Animate the pedaling so the legs follow the pedals, the wheels rotate consistently with movement, and the body moves in coordination. Set up the scene, lights, camera, and interactive controls in code. Return one runnable HTML file. Use no SVG, 2D canvas drawing, external models, or prerendered animation.

Compare one-shot generation with a repair loop that receives browser errors and evidence from the rendered scene. Build a harness that runs each artifact in an isolated browser, records errors, captures fixed viewpoints and pedal phases, and replays a camera interaction. Score launch success, visual recognizability, pedaling and wheel coordination, and working controls using automated checks where possible and a short blinded rubric for visual criteria. In week 1 freeze the prompt and rubric, in week 2 build the harness, in week 3 run matched-budget trials and a few held-out prompt variants, and in week 4 publish results and reproducible artifacts. See Willison's original SVG prompt and three.js documentation.

GameCraft-Bench for building playable games. Use the benchmark's 2D Godot tasks to test whether a coding agent benefits from staged planning and replay feedback. Select a small, fixed subset spanning two or three game families and compare a direct build agent, a plan-then-build agent, and a plan-build-replay-repair agent at a matched token or time budget. Measure project launch rate, implemented mechanics, held-out replay completion, and cost per playable project. Use week 1 to install the benchmark and reproduce a task, week 2 to implement the variants, week 3 to run the fixed subset, and week 4 to deliver launchable projects, replays, and a failure analysis. See the official GameCraft-Bench repository.

Machine design with a simulation feedback loop. Can an agent revise a mechanism more successfully when it sees measured constraint violations? Give it a parameterized gear pair or linkage and a brief specifying a motion ratio, clearance, and size. Let it propose dimensions, run a simulator or independent kinematic checker, and revise from the measurements. Compare a one-shot design, a model that generates independent designs at the same generation budget, and a parameter sweep. Match the number of simulator evaluations and report total runtime separately. On unseen briefs, report feasible-design rate, objective error, and calls or time to the first feasible design. Keep four weeks to one mechanism family, one checker, and a replay that shows the initial failure and revised design. The interactive rendering shows the iteration and the checker establishes performance under the stated assumptions. Demo inspiration, Airsup Machine Skill, and Physics-in-the-Loop.

A.3 Generative AI for education

Textbook passages as short English skits. Adapt one textbook passage into a 60–90 second dialogue that teaches four to six target words through context. Compare a direct script prompt with

a plan that first maps each word and factual claim to the source, then writes the dialogue. Use the same model, target words, and running time. Score factual fidelity, natural use of the words, and whether learners can use them in new sentences. Deliver two versions of the skit, transcripts, and a short transfer quiz. Two format references supplied for inspiration are the Gaoyou duck egg example and the WordsFunny example.

A math explanation video whose narration follows the diagram. Teach one concept such as why a secant slope approaches the tangent slope. Compare an unconstrained storyboard with a structured storyboard that links each narrated step to an equation and a visual event. Render both versions with the same graphics and comparable running time. Score mathematical errors and narration–diagram mismatches, then use a new graph to test whether viewers can apply the concept. Deliver two short videos, an editable storyboard, and a scoring rubric. A single curve, two movable points, and a line are enough to build the animation. See the Manim documentation.

An interactive 3D biology lesson grounded in a molecular structure. Teach students to distinguish protein chains from bound ligands and locate heme in hemoglobin. Let them rotate the same structure and ask questions such as *Which heme is associated with this chain?* Compare a model with access to the same lesson text in both conditions, with or without the viewer's selected component and structure annotations. Measure whether answers identify the correct object, cite relevant structural evidence, and help learners locate the corresponding feature on a different chain. Deliver one interactive molecule, a small set of annotated queries, and a transfer task. Reuse a hemoglobin structure and browser viewer from PDB-101 and RCSB Mol*.

A language-controlled 3D physics experiment. Teach how launch speed and angle affect projectile range at fixed gravity and equal launch and landing height. Build one scene with a sphere, ground plane, and trajectory trace. Let learners request controlled changes, predict the outcome, run the experiment, and compare the result with their prediction. Compare a prompt-only interpretation of the request with a model that issues typed commands and explains results from the simulator trace. Score valid commands, whether each experiment changes only the intended variable, and explanation accuracy on unseen settings. Use a separate transfer question to assess learning. Human evaluation can extend the project after the scripted tests are complete. Deliver one scene, runnable experiment scripts, and an evaluation set of paraphrased requests. See Web VPython.

Starter assets for a single lesson. Use PDB 4HHB for molecule selection and annotation. Build projectile motion from a sphere, ground plane, and trajectory trace, or build a pendulum from a pivot, rod, and bob. For an astronomy lesson, choose one model from NASA's 3D resources and attach one measurable interaction, such as locating a feature or comparing viewpoints. Keep the asset source and lesson assumptions in the README.

A.4 Training, model internals, and serving

Weight-space programming. Can a language model repair a specified failure by directly editing a checkpoint? Takehi et al. report early experiments reading and writing network weights. Their Qwen3.5-0.8B case allowed a bounded number of forward passes on development examples and used no gradients. Start with a tiny MLP or policy with a reproducible failure. Give the editing model read access to the actual tensor entries, shape metadata, a fixed edit budget, and a separate development set. Compare the proposed patch with the original checkpoint and a random edit matched for parameter count and norm. Add a simple task-specific repair baseline when available. Score held-out task success, retention on unaffected cases, and the size of the edit across seeds. In four weeks, deliver one reproducible patch experiment on one model family. Save the exact tensor diff and log any forward passes used during editing. A small language model checkpoint is a stretch goal. Takehi et al., Sparks of Weight-Space Programming.

Mechanistic interpretability at scale. Do sparse autoencoder (SAE) features that appear interpretable on examples have a consistent causal effect across prompt templates? Use released Gemma Scope weights to study a single behavior, such as copying a repeated sequence. Use activation examples to select features on development prompts. Intervene on held-out prompts and compare the effect with a no-op, random features at matched intervention magnitude, and a simple neuron

baseline. Report the change in target-token log odds, effects on unrelated prompts, and results by template. Within four weeks, cover one model, one or two layers, and one behavior. Compare two available SAE widths at comparable sparsity only after the intervention pipeline works. Gemma Scope and Scaling and Evaluating Sparse Autoencoders.

From scratch to a 0.5B training experiment. Under a fixed token budget, compare random sampling with a single data intervention such as deduplication, keeping architecture, tokenizer, and optimizer fixed. Start with a smaller randomly initialized model to establish the learning curves, then run both conditions at 0.5B parameters if measured throughput and available compute permit. Deliver configurations, matched checkpoints, held-out loss, and measured tokens per second. See Chinchilla.

Feedback for GPU kernel generation. On a fixed subset of KernelBench, compare independent candidates with iterative repair that receives compiler, correctness, or profiling feedback under a shared generation budget. Test correctness on fresh inputs, report compilation time separately, and benchmark on one GPU against both eager PyTorch and a compiled baseline where supported. Deliver the harness, generation traces, and the fraction of all tasks that are both correct and faster than each baseline. See KernelBench.

Serving short and long requests together. Replay a mixed workload with short interactive requests and long prompts on one model and one GPU. Vary the serving engine's batched token budget and measure median and 95th-percentile time to first token, inter-token latency, and throughput under the same arrival trace. Deliver a replay script and explain which settings meet a stated latency target at each request rate. See vLLM optimization guide.

A.5 Evidence and verification

Auditable evidence for medical or legal QA. Does verifying each answer claim against a cited source span reduce unsupported answers at an acceptable cost in coverage? Choose one dated public medical guideline collection or one legal corpus. Build a replayable pipeline that records the retrieved passage, document version, exact supporting span, and verifier decision for each claim. Compare ordinary retrieval-augmented generation with the same retriever and model plus span verification and abstention. On 50–100 held-out questions, including ambiguous or unanswerable cases, measure span recall, claim support, citation completeness, and the error rate among answered questions. Manually audit a sample. Deliver a source-inspection demo in four weeks using public evidence and offline scenarios. ALCE, LegalBench-RAG, and clinical guideline QA benchmark.

Reference links for featured projects

Data and adaptation. Synthetic data for instruction tuning, PEFT, Prefix-Tuning, and Direct Preference Optimization.

Reasoning and agents. ToolLLM, Outcome-supervised value models, Let's Verify Step by Step, and Voyager.

Spatial and multimodal understanding. LLaVA, ReferIt3D, SQA3D, and Habitat 3.0.

Systems and compression. LLM-Pruner, ShortGPT, KernelBench, and vLLM optimization.

Safety and interaction. Weak-to-Strong Jailbreaking, ReALM, and Collaborative Gym.

Domain examples. HuatuoGPT, HuatuoGPT-II, and Retrosynthetic planning.

B Replication and Benchmark Projects

Each brief adapts an exercise from a related course or benchmark to the four-week project. Begin with the smallest runnable case and keep the comparison focused on one change.

B1. Reproduce a model result and test one data change. The Tsinghua AML 2024 course archive connects implementation exercises with a team research project. Reproduce a small model result with the published configuration, then compare the original data with one deduplicated or quality-filtered variant. Keep the tokenizer, architecture, and training-token budget fixed. Deliver a run configuration, a matched learning curve, and a comparison on held-out data.

B2. Test when an assistant should ask for clarification. Adapt the interaction-centered evaluation in Stanford CS 329X. Select one setting such as 3D object selection, GUI actions, or document editing. Compare immediate execution, asking on every ambiguous case, and asking only when a defined trigger fires. Freeze the model, task instances, and information available before the decision. Measure task success, questions asked, and corrections. Deliver a replayable task set and a comparison. A small human study may extend the scripted evaluation.

B3. Build a minimal agent and add a recovery mechanism. Adapt the tool-use and harness practice from Stanford CS 329Z. Implement a small loop with one or two tools and evaluate it on tasks with controlled transient failures or invalid arguments. Compare plain retry with typed error feedback and a bounded repair policy under the same tool-call and model budget. Deliver the harness, replay logs, recovery rate, and cost per successful task.

B4. Train and profile a small language model from scratch. Adapt the implementation sequence in Stanford CS 336. Start from random weights, run a small decoder end to end, and establish a reproducible training baseline. Compare two model sizes under a matched approximate compute budget or vary one throughput setting with the model held fixed. Report held-out loss, actual tokens per second, peak memory, and the measured budget for any optional 0.5B run. Deliver configurations, checkpoints or logs, and a profiling summary.

B5. Improve a working method under a sealed evaluation. Use one public RSI-Exam task with a grader, or define a small task from a game, simulator, or retrieval workflow. Freeze a starter method, a visible development split, a sealed test split, and a resource cap. Compare independent attempts with iterative revision that retains successful changes. Run each frozen artifact in a clean environment and report development and held-out gains, cost, and one failure trace. Deliver the task bundle and a repeatable comparison. Teams developing a new task can also consult the task guidelines and optional contribution process. The organizers list December 15, 2026 for completed RSI-Exam 2.0 contributions and review task proposals before implementation.

C Competition Projects

A competition project uses a contest task, dataset, or simulator to answer a course research question. You may enter an open competition or run an archived task locally. Both routes count as final projects and use the course poster and report requirements.

1. **Register a course plan.** Identify the contest, track, task, official metric, and the change you will test. Note the applicable model, data, runtime, and submission rules.
2. **Run a controlled comparison.** Reproduce one baseline, keep a local held-out split or fixed simulator scenarios, and compare your method at a matched budget. Use the public leaderboard for an additional result when available.
3. **Submit course deliverables.** Report the local results, resource use, and error analysis. Include code and instructions to reproduce the baseline and proposed method. Add a dated competition score or submission link if you entered. Course assessment follows the regular rubric.

C.1 Chinese-language competitions

The three Tianchi opportunities below had open entry in the official listings checked on November 3, 2026. For course credit, teams should keep a fixed local evaluation set and record the configuration used for each submission.

- **Zhihu Zhixuetang Complex Table Recognition and Understanding (Tianchi).** Tianchi lists the challenge through December 31, 2026. Teams recover complex tables from images and PDFs and submit automatically scored Excel answers. Compare a vision-language model with an OCR-plus-structure pipeline. Report accuracy separately for merged cells, value extraction, and arithmetic. Competition · Official listing
- **Qoder University Creativity Competition (Tianchi).** Registration and project posting close November 30, 2026 at 18:00 Beijing time. Eligible university students compete in teams of one to three, use Qoder to build their application, deploy it on Qoder Sites, and post a project writeup. For StudyHack, compare a document assistant with and without evidence citations on a fixed set of research tasks, then demonstrate the deployed application. Competition rules
- **OpenLET Embodied Intelligence Challenge (Tianchi).** Registration continues through the competition season. The winter simulation round runs September 29 to December 31, 2026. Use the provided simulation and an existing policy on a single manipulation task. Compare the frozen policy with a small adaptation or a recovery rule and report success by object placement and perturbation. Hardware evaluation is available in later rounds. Competition and schedule
- **Multimodal Customer-Service Agent Design (DataFountain, archived).** The competition ran March 16 to August 31, 2026. Its task specification defines a text-and-image chat endpoint backed by a knowledge base. Recreate the task with documents and images the team may use. Compare text-only and image-aware retrieval and measure grounded answer accuracy and appropriate abstention. Archived specification

C.2 Kaggle and international benchmarks

Dates below were checked on November 3, 2026. Unless marked AOE (Anywhere on Earth, UTC−12), Kaggle deadlines are at 23:59 UTC. Plan the course experiments around the course deadlines. Follow the organizer schedule for any external submission.

- **Google DeepMind Gemma 4 Developer Agent.** Enter by November 25, 2026 and submit by December 2, 2026. Agents repair real repository issues, and held-out tests score the patches. Compare repository search alone with search plus a test-and-repair loop on the same issues, reporting pass rate, tool calls, and runtime. The offline scoring harness requires an approved Gemma 4 variant. The organizer has added 12B, E4B, and E2B variants alongside 31B. The separate paper track closes November 12. Competition · Model update · Paper track

- **Name That LLM.** Enter by January 6, 2027 and submit by January 13, 2027. Identify which of 18 models generated a passage. Compare character n-grams with a small encoder on held-out prompts, reporting balanced accuracy and confusion across model families. Scored notebooks run offline with a nine-hour limit. Publicly available data and pretrained models are permitted. Competition and rules
- **Pokémon TCG AI Battle Challenge Playground.** Enter by January 1, 2027 and submit by January 8, 2027. Use the simulator SDK to compare a heuristic policy with one that searches over legal actions. Report win rate against fixed opponents and time per move. The community ladder provides a separate comparison against other submissions. Verify execution restrictions before designing around a hosted model. Competition and timeline
- **BuildArena S02 Human–AI Engineering Challenge.** The organizer lists November 14, 2026 AOE as the submission date and directs entrants to the Kaggle countdown for the exact cutoff. This short window suits teams with a working prototype. Compare agent-only design with natural-language copilot guidance in a physics simulator. Deliver a controller, telemetry, and an interaction transcript that reproduce the mission. Tracks and submission rules
- **NeurIPS Smart Buildings.** Enter by January 3, 2027 and submit by January 15, 2027. Compare a fixed HVAC controller with a learned or model-assisted policy in simulation. Measure comfort, energy cost, and emissions on held-out scenarios. Scored notebooks run offline with a nine-hour limit. Competition and rules

The 2026 ARC-AGI-2 and ARC-AGI-3 prediction tracks closed on November 2 UTC. Course teams can still compare executable program search on the public ARC-AGI-2 grid tasks or test agent memory in public ARC-AGI-3 interactive environments. ARC-AGI-2 · ARC-AGI-3 · Benchmark guide

D Data, Training, and Model Adaptation

Change one data or training decision and evaluate its effect under a fixed budget.

D.1 Data & Knowledge Engineering

Scope. High-quality data pipelines for pretraining/supervision/preference data and synthetic data; cleaning, deduplication, filtering, scoring; content provenance and licensing/privacy compliance; data distillation.

Example Topics. (1) Build a reproducible web/text cleaning pipeline with deduplication and quality scoring. (2) Construct a small preference dataset and study its impact on instruction following. (3) Compare synthetic vs. curated human data for a targeted task.

Deliverables. Reproducible scripts, data cards (sources, licenses, risks), ablations and error analysis.

Primary Metrics. Data coverage/quality scores, downstream gains (accuracy, F_1), toxicity/PII leakage checks.

Further ideas. Compare deduplication policies, select difficult SFT examples, control code-to-text mixtures, curate spatial instructions, or audit train–test overlap.

D.2 Training & Alignment

Scope. Continued pretraining, SFT, preference optimization (DPO/PPO/GRPO), process supervision, and safety-aligned training; efficient adaptation to new domains/languages.

Example Topics. (1) SFT vs. SFT+preference optimization trade-offs on math/code tasks. (2) Low-resource language adaptation with small curated corpora. (3) Process rewards for stepwise reasoning.

Deliverables. Training recipes, config files, checkpoints/logs (or reproducible seeds), learning curves.

Primary Metrics. Task accuracy/Pass@k, instruction-following scores, calibration/harmlessness, data/compute cost.

Further ideas. Train a small language model from random initialization, compare data curricula, allocate a fixed LoRA budget across layers, or measure forgetting after domain adaptation.

E AGENTS, REASONING, AND KNOWLEDGE SYSTEMS

Change how a model searches, retrieves knowledge, plans, or calls tools. Evaluate the resulting behavior on the same tasks.

E.1 REASONING, TOOLS & AGENTS

Scope. Inference-time enhancement and closed-loop task solving: tool use, code/web/desktop agents, planning, memory, self-critique, verification.

E.1.1 TEST-TIME SCALING

Goal. Improve solution quality by allocating more compute at inference time at a stated latency and cost budget.

Typical Methods. Diverse thinking with self-consistency, reflection/reflexion loops, solution+verifier dual tracks, search-based reasoning (beam/tree-of-thought), evidence sampling & self-checking.

Suggested Tasks. Math/logic (e.g., grade-school to olympiad level), code generation/repair, knowledge QA with citations.

Deliverables. A minimal reproducible benchmark harness that sweeps "thinking steps"/"verifier strength", and reports the quality–latency–cost frontier.

Primary Metrics. Accuracy/Pass@k vs. latency and token/compute cost; robustness across seeds; reproducibility.

E.1.2 AGENT PLANNING AND TOOL USE

Goal. Build systems that plan, call tools/environments, maintain memory, and verify/repair outputs to accomplish open-ended tasks.

Suggested Benchmarks. Web/browsing tasks, desktop/OS automation, code repair (SWE-style tasks), structured interaction environments.

Deliverables. End-to-end reproducible pipelines with logs/replay, sandboxing, and failure-recovery cases.

Primary Metrics. Task success rate, retries/rollbacks, robustness to DOM/UI changes and prompt injection, side effects/safety incident rate, total cost.

Further ideas. Compare direct grid prediction with program search on ARC, allocate search budget across tasks, or compare full replanning with local plan repair.

E.2 KNOWLEDGE-INTEGRATED SYSTEMS (RAG & BEYOND)

Scope. Retrieval-augmented generation (RAG), graph-structured retrieval (GraphRAG), and workflow-style Agentic RAG with planning, citation repair, and verification; regressions & observability for enterprise use.

Classical RAG. Chunking, retrieval, reranking, context assembly, citation formatting; latency/quality trade-offs.

Agentic RAG. *Query Planner* (decomposition), *Retriever Ensemble*, *Draft & Critic*, *Citation Repair*, *Regression Suite* (gold Q/A with versioned corpora).

Deliverables. A reproducible retrieval pipeline and a small regression suite for retrieval quality, groundedness, and citation validity.

Primary Metrics. Groundedness, citation correctness, regression pass rate, and end-to-end latency.

Further ideas. Study retrieval triggers, conflicting sources, stale documents, memory compression, or per-character event memory for a virtual NPC.

F Multimodal and Human-AI Interaction

Choose one modality, interaction mechanism, or embodied task. Measure task success and the effort required from the user or agent.

F.1 Multimodal & Embodied

Scope. Vision/Audio/Video/EEG understanding & generation; speech-to-speech assistants; VLA (Vision-Language-Action) and high-level robot planning in a fixed environment.

Example Topics. (1) Real-time speech assistant with barge-in and memory. (2) Small VLA manipulation task (pick-and-place) with transfer. (3) Video captioning/editing with quality controls.

Deliverables. Demo scripts, latency breakdowns, instruction sets, task success analyses.

Primary Metrics. Task success rate, end-to-end latency, alignment with references (BLEU/METEOR/BERTScore or task-specific).

Further ideas. Resolve viewpoint-dependent 3D references, answer questions with geometric evidence, choose a new camera view, or study adaptive screenshot cropping.

F.2 HCI & Applications

Scope. Interaction design and usability; multi-agent social simulation (class forums, AI towns); education/games/vertical prototypes with real users or realistic user studies.

Example Topics. (1) Memory/Reflection designs in an educational assistant and their effect on satisfaction/learning gains. (2) AI town/forum simulation to study group dynamics and moderation. (3) Task-oriented UX for grounded report writing with inline citations.

Deliverables. A runnable prototype, a task protocol, and interaction logs or a replayable simulation. Human user studies are optional.

Primary Metrics. Task completion time, SUS/UMUX scores, error rate, qualitative feedback, retention.

Further ideas. Study when an assistant should ask a clarification question, how users interrupt a task, or how pointing resolves ambiguity in a 3D editor.

G Efficiency, Evaluation, and Reliability

Study a systems trade-off or develop an evaluation that can detect a specific failure. Fix the workload and measurement protocol before comparing methods.

G.1 Systems, Efficiency & Edge

Scope. Serving/inference engines (e.g., paged attention), quantization/distillation, KV-cache, speculative decoding; throughput–latency–energy trade-offs; on-device/edge deployment.

Example Topics. (1) Engine comparison under identical prompts (throughput, p50/p95 latency, energy). (2) Quantization + distillation impact on accuracy. (3) Edge real-time pipeline with limited memory/compute.

Deliverables. Bench harness, profiles, energy/latency curves, tuning guidelines.

Primary Metrics. Tokens/s, latency distribution, Joules/token (or power), accuracy deltas, cost.

Further ideas. Compare kernel generation feedback, quantization backends, prefix cache reuse, speculative decoding, or scheduling for mixed prompt lengths.

G.2 Evaluation, Safety & Governance

Scope. Comprehensive capability evaluation (instruction-following, dialogue, long-context), red-teaming (jailbreak, prompt injection, data exfiltration), transparency and compliance (disclosures, provenance).

Example Topics. (1) Long-context reasoning and long-form generation quality. (2) Combined defenses (input sanitization + sandbox + policy model) against jailbreak/prompt injection. (3) Risk/benefit and disclosure design in a regulated application.

Deliverables. Evaluation suites with seeds and replays, threat models, policy docs/checklists.

Primary Metrics. Task scores, robustness under perturbations, false-positive/false-negative trade-offs, compliance coverage, reproducibility.

Evaluation of reasoning and tool use. Report quality, latency, and cost under a common budget. Keep replayable intermediate logs and measure robustness to perturbations relevant to the task.

Further ideas. Check scene constraints and collisions, test unsupported success claims, audit verifier bias, or measure robustness under controlled input perturbations.