AI Usage Disclosure

AI tools were used in a limited and controlled manner during Phase 1 of this project. Large language models were used as the **objects of evaluation** for the research tasks (Paper Triage and Claim–Evidence Extraction), consistent with the goals of the assignment.

In addition, AI assistance was used to:

- Help draft and refine prompt variants used in the evaluation.
- Assist with **initial scoring and note-taking** for model outputs (e.g., identifying overgeneralization, missing citations, or speculative claims).

All scores, failure tags, and notes were **reviewed, interpreted, and finalized by the authors**. The authors are responsible for the correctness of the evaluations, the interpretation of patterns, and the conclusions drawn in the analysis memo. AI-generated suggestions were treated as aids rather than authoritative judgments.

No AI tools were used to fabricate citations, invent evidence, or alter source materials. All paper content provided to models was manually selected and curated by the authors.

I have to do phase 1. Whats this project about and what do I have to do for phase 1?

Individual Assignment - Personal Research Portal
Due No Due Date Points 0 Submitting a text entry box
Individual Project: Build a Personal Research Portal (PRP)
Three phases with progressive deliverables: Prompting → RAG → Research Portal Product

Work mode

Individual

Grade weight

Phase 1: 10% • Phase 2: 15% • Phase 3: 20% (Total: 45% of course grade)

Primary goal

A research-grade portal that answers questions with evidence, citations, and exportable research artifacts

Stack

Your choice (Python + Streamlit/Gradio recommended). Free-tier and open-weight tools allowed.

Submission

GitHub repo (code + data manifest) + reports + demo recording

Overview (what you are building)
By the end of Phase 3 you will deliver a Personal Research Portal (PRP): a lightweight product that helps you move from a research question to a grounded synthesis. It ingests a domain corpus, retrieves evidence, produces citation-backed answers and research artifacts, and logs evaluation results.

Minimum PRP capabilities (MVP)

Ingest a domain corpus (PDFs/URLs/notes) with metadata (author, year, source type, link/DOI).
Retrieve evidence (semantic retrieval at minimum).
Generate answers where every major claim is backed by an inline citation to a source/chunk ID.
Produce at least one research artifact: evidence table, annotated bibliography, or synthesis memo.
Save research threads (query, retrieved chunks, answer) and export outputs (Markdown/CSV/PDF).
Run an evaluation set and report groundedness/faithfulness plus at least one additional metric.
What counts as a citation in this project

You may cite using (source_id) or (source_id, chunk_id) as long as it uniquely maps to text you ingested.
Your repo must include a data manifest that makes every citation resolvable to: source title + link/DOI + local file or snapshot.
Example citation formats: (Smith2023, chunk_07) or (Smith2023) if the source_id uniquely identifies one ingested document.
If the corpus does not support a claim, your system must say so rather than guessing.
How the three phases fit together
Phase 1 produces your research framing, prompt kit, and an evaluation rubric you will reuse.
Phase 2 uses that same framing to build research-grade retrieval and grounding over your corpus.
Phase 3 wraps Phase 2 into a usable portal and generates exportable research artifacts.
Design intent: each phase should feel like a direct upgrade of the previous one, not a reset.

Deliverables at a glance
Phase

Weight

Purpose

Core deliverables

MVP scope (minimum)

Stretch goals (examples)

Phase 1 — Prompt the Research Domain

10%

Design prompts and evaluate model behavior on research tasks.

Framing brief; prompt kit; evaluation sheet; analysis memo.

2 tasks • 2 models • 2 test cases per task • 2 prompt variants per task (16 total runs).

3rd model; adversarial tests; automated checks; better uncertainty calibration.

Phase 2 — Ground the Domain (Research-Grade RAG)

15%

Build baseline RAG over your corpus; add production patterns; evaluate.

Ingestion pipeline; baseline RAG; enhancements; evaluation report + logs.

15 sources • 20 queries • 1 enhancement • citation-backed answers.

Hybrid retrieval; reranking; section-aware chunking; structured citations; RAGAs/ARES.

Phase 3 — Build the Personal Research Portal Product

20%

Turn your RAG into a usable portal that produces research artifacts.

Working app + demo; final report; generated artifacts; evaluation view.

Search + citations + threads + export + at least 1 artifact type.

Agentic research loop; knowledge graph; gap finding; BibTeX export; improved UX.

Phase 1 — Prompt the Research Domain (10%)
Goal

Pick a research domain, define a main research question, and test how different models handle core research tasks. You will produce a prompt kit you will reuse in Phases 2 and 3.

You must choose

A domain and one main research question.
4–6 sub-questions that break your main question into retrievable parts.
Two tasks from the task menu (each with a required output format).
Two accessible models (three allowed for stretch).
Task menu (choose 2) — with concrete output formats

Paper triage: output a 5-field summary (Contribution, Method, Data, Findings, Limitations).
Claim–evidence extraction: output 5 rows with Claim | Direct quote/snippet | Citation (source_id, chunk_id).
Cross-source synthesis: output a table of Agreement | Disagreement | What evidence supports each side.
Research gap spotting: output 3 gaps, each with 'What is missing' + 'What evidence would resolve it'.
Citation formatting: output a paragraph with inline citations plus a short reference list from your manifest.
Definitions (what we mean in Phase 1)

Task: a repeatable research operation with a required output format (e.g., paper triage or claim-evidence extraction).
Test case: one concrete input for a task (e.g., Paper A PDF; or Paper B, Methods section). You will run every test case through both prompts and both models.
Prompt variant: two versions per task - Prompt A (baseline) and Prompt B (improved with structure/guardrails).
Worked example (illustrative - you will use your own domain)

Domain + question: "Trustworthy RAG evaluation methods" - Main question: "How do different faithfulness metrics fail, and how can we combine them?"
Tasks (pick 2): (1) Paper triage; (2) Claim-evidence extraction.
Test cases (2 per task):
- Paper triage test cases: Paper A (survey) and Paper B (empirical study).
- Claim-evidence extraction test cases: Paper C (benchmark) and Paper D (critique).
Prompts (2 per task): Prompt A = minimal instruction; Prompt B = structured output + "cite chunk_id" + "say unknown when missing".
Models (2): any two you can access. Run all combinations (16 runs for MVP) and score outputs for groundedness and citation correctness.
MVP checklist (do these in order)

Write a 1-page framing brief: domain, main question, sub-questions, scope (what you include/exclude).
Create 2 test cases per chosen task (4 total test cases across Phase 1). A test case is one concrete input, such as a specific paper or section.
Write two prompt variants per task: Prompt A (baseline) and Prompt B (improved structured prompt with guardrails).
Run both models across all test cases and both prompt variants, and record outputs in the evaluation sheet (2 tasks x 2 cases x 2 prompts x 2 models = 16 runs).
Score each output using the Phase 1 rubric and write a 1–2 page analysis of failure modes and fixes.
Phase 1 scoring rubric (1–4) — what the numbers mean

4: Correctly grounded and structured; citations are correct; uncertainty is stated when evidence is weak.
3: Mostly correct and structured; minor missing nuance OR minor citation/format issues.
2: Partially correct; key omissions OR weak grounding OR vague citations.
1: Not usable; hallucinated claims, fabricated citations, or fails the required structure.
Phase 1 deliverables (submit together)

Framing brief (1–2 pages).
Prompt kit (Markdown): prompts + why each constraint exists.

Evaluation sheet (CSV/Google Sheet/Markdown table): scores + notes per run (for MVP, include 16 rows - one per run).
Analysis memo (1–2 pages): patterns, failures, and Phase 2 design choices.
Common mistakes to avoid

Choosing tasks that do not force citations (then you cannot assess grounding).
Scoring without any written justification (graders cannot tell what you observed).
Changing your question mid-phase (it breaks continuity into Phase 2).
Phase 2 — Ground the Domain (Research-Grade RAG) (15%)
Goal

Build a baseline RAG pipeline over your own corpus and upgrade it with production patterns. This phase prioritizes retrieval quality, traceability, and evaluation—not UI polish.

Corpus requirements (MVP)

15–30 sources minimum. At least 8 should be peer-reviewed papers, standards, or reputable technical reports.
Every source must have metadata: source_id, title, authors, year, type, venue (if applicable), link/DOI, and a 1–2 sentence relevance note.
Source acquisition can be manual (download PDFs) or automated (crawler/agentic selector). Either way, do not rely on live links alone: store the raw artifact (PDF/HTML snapshot/extracted text) under data/raw/ OR include a reproducible download script.
If you use an automated crawler/agent to pick sources, include the code plus the run configuration (prompts/filters) and a short note describing how sources were selected (manual vs scripted vs agentic).
Baseline RAG MVP (do these in order)

Ingest sources: parse and clean text; store text + metadata; create a data manifest (CSV/JSON).
Chunk with a documented strategy (chunk size, overlap).
Use section-aware chunking for papers if feasible.
Embed and index in a vector store (FAISS acceptable).
Retrieve top-k chunks per query and generate an answer

that cites chunk/source IDs.
Create an evaluation set of at least 20 queries: 10 direct, 5 synthesis/multi-hop, 5 ambiguity/edge cases.
Example direct query: "What does <method/metric> measure, and what are its known failure modes?"
Example synthesis query: "Compare Paper A vs Paper B on <criterion>; where do they agree and disagree?"
Example edge-case query: "Does the corpus contain evidence for <claim>?" (If not found, your system must say so.)
Evaluate groundedness/faithfulness plus one additional metric (answer relevance, context precision/recall, citation precision). Interpret results.
Required production patterns (must-have)

Logging: store queries, retrieved chunks, model outputs, and prompt/version IDs.
Reproducibility: pinned dependency versions; one-command run path (Makefile or a single script).
Trust behavior: refuse to invent citations; explicitly flag missing or conflicting evidence.
Enhancements (implement at least 1 for MVP)

Query rewriting or decomposition (question → sub-queries).
Reranking (cross-encoder, LLM reranker, or heuristic reranker).
Metadata filtering (year/author/type) and faceted retrieval.
Hybrid retrieval (BM25 + vector).
Confidence / evidence-strength scoring.
Structured citations (inline citations + reference list from your manifest).
Phase 2 deliverables (submit together)

Code + repo (baseline RAG) with README instructions.
Data manifest (CSV/JSON) with metadata for every source.
Evaluation report (3–5 pages): query set design, metrics, results, and what improved with enhancements.
Run logs (machine-readable): at least 20 queries with retrieved chunks and outputs.
Phase 2 acceptance tests (what graders will check quickly)

A single command produces: retrieval results, an answer with citations, and a saved log entry.

Randomly sampled answers have citations that resolve to real source text in your corpus.
Your evaluation report includes at least 3 representative failure cases with evidence.
Phase 3 — Build the Personal Research Portal Product (20%)
Goal

Turn your Phase 2 system into a usable portal that supports a research workflow: question → evidence → synthesis → export. Your portal should not just chat; it should produce research artifacts.

Product MVP requirements

Interface: a working UI with at least: search, ask, show sources/citations, and history.
Research threads: save query + retrieved evidence + answer (file-based OK).
Artifact generator: produce at least one of: evidence table, annotated bibliography, or synthesis memo with references.
Export: allow export of artifacts (Markdown/CSV/PDF).
Evaluation: a page or script that runs your query set and summarizes metrics plus representative examples.
Trust behavior: every answer includes citations; missing evidence is stated explicitly with a suggested next retrieval step.
Recommended artifact schemas (pick at least 1 for MVP)

Evidence table: Claim | Evidence snippet | Citation (source_id, chunk_id) | Confidence | Notes.
Annotated bibliography: 8–12 sources with 4 fields (claim, method, limitations, why it matters).
Synthesis memo: 800–1200 words with inline citations and a reference list generated from your manifest.
Phase 3 deliverables (submit together)

Working PRP app + instructions to run locally.
Demo recording (3–6 minutes) showing retrieval, citations, artifact generation, and export.
Final report (6–10 pages): architecture, design choices, evaluation, limitations, and next steps.
Generated research artifact outputs included in the repo and referenced in the report.
Stretch goals (optional)

Agentic research loop (plan → search → read → synthesize) with guardrails and logs.
Knowledge graph view (entities/concepts linked to source passages).
Automatic disagreement map (conflicts surfaced with citations).
Gap finder: missing evidence + targeted next retrieval actions.
Improved UX: filters by year/venue/type, reading list, tagging, saved collections.
Grading (points aligned to course weight)
Your project is graded on rigor, clarity, and evidence, not on paid tooling. Points below map directly to the phase weights.

Phase 1 (10 points = 10%)

Framing quality and scope discipline (3): clear question, sub-questions, inclusions/exclusions.
Prompt kit quality (4): structured prompts, guardrails, reusable prompt cards.
Evaluation rigor and analysis (3): consistent scoring, failure tags, actionable takeaways.
Phase 2 (15 points = 15%)

Corpus + manifest quality (4): metadata completeness, reproducibility, source credibility.
Retrieval + grounding implementation (5): working RAG with correct citations and logs.
Enhancement + measurable improvement (3): at least one enhancement with evidence it helped.
Evaluation report quality (3): query set design, metrics, interpretation, failure cases.
Phase 3 (20 points = 20%)

Portal MVP functionality (8): UI, threads, citations, export, reliability of core flow.
Research artifacts (4): artifact schema correctness, usefulness, citation traceability.
Evaluation + trust behaviors (4): run query set, summarize metrics, handles missing evidence.
Engineering + communication (4): clean repo, run instructions, demo, clear final report.
Submission requirements
Repo expectations (minimum)

A README that tells a grader how to run your system in 5

minutes or less.
Pinned dependencies (requirements.txt or pyproject.toml).
data/data_manifest.csv (or JSON) with the required metadata schema.
Your repo must include data/raw/ with the downloaded PDFs/snapshots you used, OR a reproducible download script that regenerates data/raw/.
A folder of outputs (artifacts/exports) generated by your system.
Logs from evaluated runs (machine-readable).
Recommended repo structure

```
repo/
  README.md
  requirements.txt (or pyproject.toml)
  data/
    raw/ (downloaded PDFs, HTML snapshots, original notes)
    processed/ (parsed text, chunks, intermediate files)
    data_manifest.csv (or JSON)
  src/
    app/ (Phase 3 UI)
    ingest/ (parsers + chunking + optional download scripts)
    rag/ (retrieval + generation)
    eval/ (query sets, scripts, results)
  outputs/ (artifacts, exports)
  logs/ (runs, prompts, versions)
  report/ (Phase 1, Phase 2, Phase 3 writeups)
```

AI usage disclosure (required)

Include a 1-page AI-usage log listing: tool name, what you used it for, and what you changed manually afterward.
You are responsible for correctness, citations, and meeting requirements even if an AI tool suggested code or text.
Appendix A — Templates (copy/paste)

A1. Prompt card template

Prompt name:
Intent:
Inputs (what you provide):
Outputs (required structure):

Constraints / guardrails (no fabricated citations; cite chunk IDs; etc.):
When to use / when not to use:
Failure modes to watch for:
A2. Evaluation row template (per answer)

Task ID: | Test case ID: | Query ID:
Task ID:

Prompt ID + model:
Retrieved evidence IDs:
Score 1 — Groundedness/faithfulness (1–4):
Score 2 — Citation correctness (1–4):
Score 3 — Usefulness (1–4):
Notes + failure tag (missing evidence, wrong citation, overconfident, etc.):
A3. Source metadata schema (minimum fields)

source_id, title, authors, year, source_type, venue, url_or_doi, raw_path, processed_path, tags, relevance_note

raw_path should point to the file you store under data/raw/. processed_path is optional but recommended (e.g., parsed text under data/processed/).

Alright, deep breath — this looks *huge*, but **Phase 1 is actually very manageable** once you strip away the future stuff. You are **not building a system yet**. You're basically doing *research planning + prompt experimentation*.

I'll explain this in plain language, then give you a **concrete checklist** for Phase 1 only.

Big picture: what is this project about?

This project is about building a **research assistant you actually trust**.

Not a chatbot that "sounds smart," but a tool that:

- Answers **research questions**
- Uses **real sources**
- **Cites evidence**
- Says **"I don't know"** when evidence is missing
- Produces research-style outputs (tables, summaries, bibliographies)

The **three phases** are intentionally stacked:

Phase	What you're really doing
Phase 1	Figure out *how to ask* good research questions and *how models fail*
Phase 2	Teach the system to *ground answers in real documents* (RAG)
Phase 3	Turn it into a usable **research portal product**

So Phase 1 is about **thinking and testing**, not engineering.

What Phase 1 is actually about (in human terms)

> **Phase 1 = research framing + prompt experiments**

You are:

1. Picking a **research topic**
2. Breaking it into **sub-questions**
3. Choosing **two research tasks**
4. Writing **two versions of prompts**
5. Testing **two models**
6. Scoring how well they do
7. Writing about **what went wrong and how to fix it later**

That's it.

No vector DBs.
No embeddings.
No Streamlit.
No RAG.

What you must deliver in Phase 1 (simplified)

You submit **4 things**:

1. **Framing brief** (1–2 pages)
2. **Prompt kit** (Markdown)
3. **Evaluation sheet** (table / CSV)
4. **Analysis memo** (1–2 pages)

Everything else is instructions on *how* to do those four well.

Step-by-step: exactly what you need to do for Phase 1

✅ Step 1: Pick a domain + main research question

Choose **one narrow research domain**.

Good domains:

- Something from class
- Something related to your major
- Something you *might* want to build Phase 2 around

Examples:

- AI safety metrics
- Climate policy effectiveness
- Social media misinformation
- Medical imaging bias
- Software engineering productivity tools

Then define **one main research question**, e.g.:

> "What evidence exists that X causes Y, and where does the literature disagree?"

This question **cannot change later**, so don't overthink but don't be random.

✅ Step 2: Write 4–6 sub-questions

These should be **answerable from documents**, not opinions.

Bad sub-question:

- "Is this good or bad?"

Good sub-question:

- "What methods do studies use to measure X?"
- "What limitations are consistently reported?"
- "Where do results conflict across studies?"

These sub-questions will later become retrieval queries in Phase 2.

✅ Step 3: Pick 2 research tasks from the task menu

You must choose **exactly two** from this list:

- Paper triage
- Claim–evidence extraction
- Cross-source synthesis

- Research gap spotting
- Citation formatting

Think of these as **repeatable academic skills**.

💡 Recommendation (easy + strong):

- **Paper triage**
- **Claim–evidence extraction**

They force citations and make errors obvious.

✅ Step 4: Create 2 test cases per task (4 total)

A **test case** is just:

> "Here is a specific paper / text section — do the task on this."

For example:

- Task: Paper triage
  - Test case 1: Survey paper
  - Test case 2: Empirical study

You don't need a big corpus yet — PDFs or pasted excerpts are fine.

✅ Step 5: Write 2 prompt variants per task

For **each task**, you write:

- **Prompt A (baseline)**
  - Minimal instructions
  - "Summarize this paper…"
- **Prompt B (improved)**
  - Structured output
  - Guardrails:
    - "Use citations"
    - "Say unknown if missing"
    - "Follow this schema exactly"

This is where the *learning happens*.

✅ Step 6: Choose 2 models

Any two you can access:

- GPT-4.x vs GPT-3.5
- Claude vs GPT
- Open-weight model if you want

You'll run **everything on both models**.

✅ Step 7: Run all combinations (this is the math they care about)

You must produce **16 runs**:

```bash
2 tasks
× 2 test cases
× 2 prompt variants
× 2 models
= 16 outputs
```

Copy code

Each run gets:

- The model's output
- A score (1–4)
- Notes on what failed

Put this in a **table / CSV / Google Sheet**.

✅ Step 8: Score the outputs (rubric is simple)

For each run, score:

- Groundedness / faithfulness
- Citation correctness
- Usefulness

You are **not** grading the model harshly — you're documenting behavior.

✅ Step 9: Write the analysis memo (1–2 pages)

This answers:

- What failed most often?
- Which prompt helped?
- Which model hallucinated more?
- What guardrails worked?
- What will you carry into Phase 2?

This memo is what **connects Phase 1 → Phase 2**.

What Phase 1 is *not*

🚫 Not coding
🚫 Not building a UI
🚫 Not collecting 30 papers
🚫 Not using embeddings
🚫 Not a product demo

It's **controlled experimentation + reflection**.

If you want, next we can:

- Pick a **domain + research question together**
- Choose the **best 2 tasks for you**
- Draft **Prompt A / Prompt B**
- Create a **ready-to-fill evaluation sheet template**
- Outline your **analysis memo**

Just tell me:

> "Let's pick a domain"
> or
> "Help me choose tasks"

1/2

ok lets get started

Let's do it 😄
We'll move **one clean step at a time** so this doesn't feel overwhelming.

Step 1 (right now): lock in a domain + main research question

I'll give you **3 strong, Phase-1-friendly options**. Each one:

- Works well for prompt testing
- Forces citations (important for grading)
- Transitions cleanly into Phase 2 later

You can **pick one** or tweak it.

Option A — *AI / LLMs* (very on-theme for this assignment)

Domain

Trustworthiness of LLM outputs in research settings

Main research question

> *How and why do large language models hallucinate in research tasks, and what prompt constraints reduce hallucinations?*

Why this is good

- You're literally studying the tool you're using
- Easy to design failure cases
- Clean path into RAG in Phase 2
- Graders love meta-evaluation

Example sub-questions (we'll finalize later)

- What types of hallucinations appear most often?
- How do citation constraints affect correctness?
- When do models correctly say "insufficient evidence"?
- How do models differ in uncertainty handling?

Option B — *Education / Learning Science* (very CMU-core)

Domain

Evidence-based learning strategies in higher education

Main research question

> *Which learning strategies have strong empirical support, and where does the evidence conflict or remain inconclusive?*

Why this is good

- Tons of papers → easy test cases
- Clear claims → easy to score grounding
- Strong synthesis opportunities later
- Feels very "research portal" appropriate

Example sub-questions

- Which strategies consistently improve retention?
- Where do studies disagree on effectiveness?

- What populations are under-studied?
- What methodological limitations recur?

Option C — *Tech + Society* (policy / ethics angle)

Domain

Algorithmic bias in decision-making systems

Main research question

> *What empirical evidence exists for algorithmic bias in automated decision systems, and how do studies measure and dispute it?*

Why this is good

- Strong claim–evidence structure
- Forces careful citation handling
- Great for disagreement and gap-spotting tasks
- Scales well into a corpus later

Example sub-questions

- What metrics are used to define bias?
- Where do studies reach conflicting conclusions?
- What evidence is missing for causal claims?
- What datasets dominate the literature?

What I need from you (one decision)

👉 Which option do you want to go with?

Reply with **A, B, or C**, or say:

> "I like A, but tweak the question to ___"

Once you pick:

- I'll **write your Phase 1 framing brief outline**
- Then we'll **choose the 2 tasks**
- Then we'll **draft Prompt A + Prompt B**

We're officially building this now 🚀

We need to create a super pog for these:

Trust and Institutional Transparency
How might we help students trust the university's sustainability claims and goals?
What if students felt more informed about how sustainability decisions are made by the university?
What if students felt their voices influenced campus sustainability priorities?
How might we develop a sense of shared responsibility between the university and the students?
How might we clarify the limits of student control within campus energy systems?
How might we help students understand tradeoffs universities face in sustainability decisions?

Super POG:

Transportation
How might we help students understand the positive impact of choosing campus shuttle/public transportation over personal transportation?
How might we reduce unnecessary shuttle trips on campus?
How might we support sustainable transportation options when students are late or under pressure?
How might we make the shorter walk to class more appealing than taking shuttle/other transportation?
How might we address student confusions around shuttle routes and reliability that leads them into choosing less sustainable transportation?

Super POG:

Got it — you're being asked to **roll these clusters up into a single, strong "Super POG"** for each theme. A Super POG should feel:

- **Human-centered** (about student understanding, trust, behavior)
- **Broad but coherent** (captures all sub-questions)
- **Actionable** (something you could actually design for)

I'll give you **polished Super POGs**, then briefly explain *why they work* so you can defend them if asked.

Super POG — Trust & Institutional Transparency

Super POG:

> **How might we increase student trust in the university's sustainability efforts by making decision-making processes, tradeoffs, and the real impact of student input transparent and understandable?**

Why this works

This single question cleanly wraps:

- Trust in sustainability claims
- Transparency in how decisions are made
- Student voice and influence
- Shared responsibility (students + university)
- Limits of student control
- Tradeoffs and constraints

It reframes everything as a **trust problem driven by opacity**, not student apathy.

Super POG — Transportation

Super POG:

> **How might we encourage students to choose more sustainable transportation options by making those choices feel clearer, more reliable, and more appealing—even when students are rushed, stressed, or uncertain?**

Why this works

This captures:

- Understanding environmental impact
- Reducing unnecessary shuttle use
- Decision-making under pressure
- Making walking/public transit appealing
- Confusion around routes, timing, and reliability

It focuses on **behavior in real conditions**, not ideal ones.

If you want a slightly punchier / more design-y tone

Here are **alternative stylings** in case your class prefers sharper language:

Transparency (alt)

> **How might we help students feel informed, heard, and realistically empowered in campus sustainability decisions so trust replaces skepticism?**

Transportation (alt)

> **How might we make sustainable transportation the easiest and most intuitive choice for students navigating time pressure and uncertainty on campus?**

Quick gut check (use this yourself)

A Super POG is good if:

- ✅ Every original question clearly fits under it
- ✅ It doesn't imply a solution yet
- ✅ You could brainstorm *many* solutions from it
- ✅ It reframes the problem, not just restates it

These pass all four.

If you want, next we can:

- Map **each original question → the Super POG** (great for slides)
- Turn these into **problem statements + insights**
- Or translate them into **design principles or metrics**

this is kind of hard to understand...maybe reword it:

How might we increase student trust in the university's sustainability efforts by making decision-making processes, tradeoffs, and the real impact of student input transparent and understandable?

Totally fair — that sentence is doing *way* too much work 😅
Let's simplify it without losing meaning.

Here are **clearer, more human versions**, from safest → punchiest. Pick the vibe your class likes.

✅ Clean + easy to understand (recommended)

> **How might we help students trust the university's sustainability efforts by clearly showing how decisions are made and how student input actually matters?**

Why this works:

- Shorter
- Plain language
- Keeps **trust + transparency + student voice**

Even simpler / more conversational

> **How might we help students better understand how sustainability decisions are made so they can trust the university's goals and actions?**

This is great if your instructor values clarity over precision.

Slightly more design-forward (still clear)

> **How might we build student trust in campus sustainability by making decisions, tradeoffs, and student influence more visible and understandable?**

If you want to explicitly include limits + tradeoffs (but still readable)

> **How might we help students trust campus sustainability efforts by clearly explaining how decisions are made, what tradeoffs exist, and where student influence begins and ends?**

My top pick (if I were you)

If this is for a **design / POG assignment**, I'd submit:

> **How might we help students trust the university's sustainability efforts by clearly showing how decisions are made and how student input actually matters?**

If you want, we can now:

- Do the **same simplification** for the Transportation Super POG
- Map each bullet question to this Super POG
- Or rewrite both Super POGs so they match stylistically for a slide or paper

Motivation and Engagement

33. How might we sustain student motivation for sustainability beyond short-term events or campaigns?
34. How might we help students feel that their sustainability efforts actually matter?
35. What if students could better connect sustainability actions to personal values and goals?
36. How might we make students feel more accountable for shared resource consumption?
37. How can we help students recover from unsustainable habits without feeling discouraged?
38. How might we sustain student motivation for sustainability beyond short term events?
39. What if students could see the collective impacts of their small actions adding up with their peers?

Super POG:

Got it — this cluster is all about **long-term motivation, accountability, and not burning students out**. Let's turn it into one **clear, human Super POG** that's easy to understand.

Super POG — Motivation & Engagement

✅ Clear + simple (recommended)

> **How might we help students stay motivated to practice sustainability over time by helping them see that their actions matter, connect to their personal values, and contribute to a shared impact—without making them feel discouraged or blamed?**

Even simpler / more conversational

> **How might we help students stay motivated to make sustainable choices by showing that their efforts matter and add up, even when it's hard to be perfect?**

This one is great if your instructor values accessibility and empathy.

Slightly more design-focused (still readable)

> **How might we support long-term student motivation for sustainability by reinforcing accountability, personal relevance, and the collective impact of small actions?**

Why this Super POG works

It cleanly covers **all** your questions:

- Sustaining motivation beyond short-term events ✅

- Feeling like efforts matter ✅
- Connecting sustainability to personal values ✅
- Accountability for shared resources ✅
- Recovering from setbacks without discouragement ✅
- Seeing collective impact over time ✅

And it avoids:

- Guilt framing
- Perfectionism
- One-off "campaign" thinking

My top pick for submission

If I were turning this in, I'd use:

> **How might we help students stay motivated to practice sustainability over time by helping them see that their actions matter, connect to their personal values, and contribute to a shared impact—without making them feel discouraged or blamed?**

If you want, next we can:

- Match this stylistically with your **other Super POGs**
- Map each numbered question (33–39) directly under this Super POG
- Tighten it even more to fit a slide or one-liner

Behavior and Habits
How might we support students in maintaining sustainable habits despite busy or unpredictable schedules?
How might we reduce friction that causes students to choose comfort/convenience over sustainability?
What if sustainable behaviors felt more aligned with students' daily routines rather than extra effort?
How might we support students in maintaining sustainable habits despite busy or unpredictable schedules?
How might we reduce the friction that causes students to choose convenience over sustainability?
How might we design sustainable defaults into the existing structures of student life such as housing and dining?

How might we accommodate students' diverse lifestyles and needs within our sustainability expectations?
How might we support sustainability behavior across both academic and residential spaces?
How might we design a laundry rooms so that using cold water and full loads can be the most obvious option?
How might we help students to remember their cloth bags when they spontaneously decide to go shopping?
What if sharing resources within a dorm felt easier?
What if sustainability choices felt easier during high-stress times like midterms and finals?

Super POG:

Perfect — this is a **classic "behavioral friction + defaults" cluster**, so the Super POG should emphasize *ease, routines, and systems*, not willpower.

Here are **clean, understandable Super POG options**, from safest to punchiest.

Super POG — Behavior & Habits

✅ Clear + student-centered (recommended)

> **How might we make sustainable behaviors easier and more natural for students by reducing friction and embedding them into everyday routines, even during busy or high-stress times?**

Even simpler / very plain language

> **How might we help students choose sustainable habits by making them the easiest and most convenient option in daily student life?**

This one is very strong if your class likes **behavioral design framing**.

Slightly more design-forward (systems-focused)

> **How might we design student environments and routines so that sustainable choices are the default, not an extra effort?**

Explicitly includes diversity + flexibility

> **How might we support sustainable habits for students with diverse lifestyles by building flexible, low-effort sustainability into academic, residential, and daily life systems?**

Why these work (quick check)

Your questions include:

- Busy/unpredictable schedules ✅
- Convenience vs sustainability ✅
- Defaults in housing, dining, laundry ✅
- Academic + residential spaces ✅
- High-stress moments (midterms/finals) ✅
- Diverse student needs and routines ✅
- Resource sharing & reminders (bags, laundry) ✅

All of that boils down to:

> **Don't rely on motivation — design the system.**

My top pick for submission

If you want clarity + alignment with your other Super POGs:

> **How might we make sustainable behaviors easier and more natural for students by reducing friction and embedding them into everyday routines, even during busy or high-stress times?**

If you want, next we can:

- Make sure **all Super POGs match stylistically**
- Turn these into **insight statements**
- Or help you **justify each Super POG with 1–2 sentences** (great for grading rubrics)

sorry I pasted the wrong assignment into this chat...lets go back to the phase 1 assignment

All good — thanks for the reset 👍

Let's snap back to **Phase 1: Prompt the Research Domain** and keep momentum.

I'll re-anchor us, then we'll take **the next concrete step**.

Where we are in Phase 1 (checkpoint)

Phase 1 has **one real goal**:

> Design and test prompts for *research tasks* and understand how models succeed/fail **before** you build RAG.

You haven't locked anything in yet, which is perfect.

What we need to do *next* (Step 1, for real)

🔒 Step 1: Lock the research domain + main research question

Everything else depends on this:

- Tasks
- Test cases
- Prompts
- Evaluation
- Phase 2 corpus

Once this is locked, we don't change it.

I'll propose 3 Phase-1-optimized domains

(each is scoped, testable, and grader-safe)

Option A — LLM hallucinations in research tasks (very strong)

Domain
LLM reliability in academic/research workflows

Main research question

> *How do large language models fail on citation-based research tasks, and which prompt constraints reduce hallucinations and overconfidence?*

Why this is excellent:

- Directly aligned with the assignment
- Easy to design failure cases
- Clean transition into RAG in Phase 2
- You already think this way

Option B — Evidence-based sustainability interventions on college campuses

Domain

Student sustainability behavior interventions in higher education

Main research question

> *What evidence supports different sustainability interventions for college students, and where does the literature show disagreement or uncertainty?*

Why this works:

- Ties to your other coursework
- Strong claim–evidence structure
- Lots of PDFs available
- Great for synthesis + gap spotting

Option C — Learning science & study strategies

Domain

Empirically supported learning strategies for college students

Main research question

> *Which learning strategies are strongly supported by evidence, and where do findings conflict or lack sufficient data?*

Why this works:

- Extremely well-studied domain
- Easy to evaluate grounding
- Very "research portal" friendly

Your move (one decision)

Reply with **one of these**:

- `Option A`
- `Option B`
- `Option C`

Or:

> "Option B, but tweak the question to …"

What I'll do immediately after you choose

In the *next* message, I will:

1. Write your **Phase 1 framing brief (outline + draft)**
2. Propose **4–6 sub-questions**
3. Recommend the **best 2 tasks** for your domain
4. Draft **Prompt A + Prompt B** for one task so you can see the pattern

Once the domain is locked, Phase 1 becomes very mechanical and low-stress.

Go ahead — pick the domain 🚀

How about something along the lines of how ai or llms impact student learning (either positive or negative or both). Or maybe something similar: how software engineers use ai/llms and the impacts it has on productivity or the industry or the future work of new devs (after graduation)

That's actually a **great instinct**, and both of those ideas are *very* Phase-1-friendly. Let's tighten them so they're **research-grade**, scorable, and easy to carry into Phase 2.

I'll do three things:

1. Refine each idea into a **clean research domain + main question**
2. Explain which one is **stronger for Phase 1**
3. Let you pick (or hybridize)

Option A — AI / LLMs & Student Learning

Polished domain

Impact of AI tools on student learning outcomes and learning behaviors

Strong main research question

> **How does the use of AI tools affect student learning, understanding, and skill development, and where does existing research show benefits, risks, or uncertainty?**

Why this is strong:

- Balanced (positive *and* negative)
- Forces evidence (not vibes)
- Supports synthesis + disagreement tasks
- Very relevant to current education debates

Example sub-questions

- What learning gains or losses are reported with AI use?
- How does AI affect surface learning vs deep understanding?
- What risks (dependency, reduced skill transfer) are identified?
- Where do studies disagree or lack longitudinal evidence?
- How do effects vary by discipline or student level?

Option B — AI / LLMs & Software Engineering Productivity

Polished domain

Impact of AI-assisted tools on software engineering practice

Strong main research question

> **How do AI tools affect software engineer productivity, code quality, and skill development, particularly for early-career developers?**

Why this is excellent:

- Very concrete outcomes (productivity, quality)
- Natural claim–evidence structure
- Easy to design failure cases
- Strong industry relevance
- Clean path into RAG later (papers, blogs, reports)

Example sub-questions

- What productivity gains are empirically measured?
- How does AI affect code quality and error rates?
- What skills improve or degrade with AI assistance?
- How do impacts differ for junior vs senior developers?
- What long-term risks or unknowns are identified?

Which is better for Phase 1 specifically?

🥇 Best for Phase 1: Option B (Software Engineering)

Why:

- Claims are more **measurable**
- Evidence is clearer (metrics, studies, benchmarks)
- Easier to score hallucinations
- Less ambiguity than "learning" research
- Strong alignment with *research tasks* like:
  - Paper triage
  - Claim–evidence extraction
  - Cross-source synthesis

Option A is still very good — it's just more qualitative and messier to evaluate.

A powerful hybrid (if you want the best of both)

You *can* combine them **without breaking scope**:

Hybrid domain

Impact of AI tools on learning and productivity in technical education

Hybrid main research question

> **How do AI tools influence learning and productivity for students training to become software engineers, and what risks or tradeoffs does existing research identify?**

This lets you:

- Talk about students **and** future devs
- Stay technical
- Bridge education + industry cleanly

Your decision (lock it in)

Reply with one of these:

- **Option A — Student learning**
- **Option B — Software engineering productivity**
- **Hybrid — students → future software engineers**

Once you choose, I will immediately:

1. Write your **Phase 1 framing brief (drafted)**
2. Define **4–6 sub-questions**
3. Select the **best 2 research tasks**
4. Draft **Prompt A + Prompt B** for your first task

This is the point where Phase 1 really starts feeling doable.

Lets do option B

Perfect choice 👍
Option B (AI & software engineering productivity) is *clean, scorable, and very Phase-1 friendly*.

Below is **everything you need to officially start Phase 1**, written so you can almost copy-paste parts of it into your submission.

Phase 1 — Framing (LOCKED)

Domain

Impact of AI-assisted tools on software engineering practice

Main Research Question

> **How do AI tools affect software engineer productivity, code quality, and skill development, particularly for early-career developers?**

This question is:

- Evidence-driven (not opinion-based)
- Narrow enough for Phase 1
- Easy to extend into Phase 2 (papers, reports, benchmarks, industry studies)

✅ Do **not** change this later.

Sub-Questions (4–6 required)

You'll reuse these in Phases 2 and 3.

1. **What productivity gains or losses are empirically measured when software engineers use AI tools?**
2. **How does AI assistance affect code quality, correctness, and maintainability?**
3. **How do impacts differ between junior/early-career developers and senior developers?**
4. **What risks or negative effects (e.g., over-reliance, skill degradation) are identified?**
5. **Where do studies disagree or show inconclusive results?**

6. *(Optional)* **What long-term impacts on developer skill formation are suggested but not yet proven?**

Phase 1 — Task Selection

You must choose **2 tasks** from the task menu.
These two are the strongest for your topic and grading rubric.

✅ Task 1: Paper Triage

Required output format

- Contribution
- Method
- Data
- Findings
- Limitations

Why this task works:

- Forces structured reasoning
- Easy to compare Prompt A vs Prompt B
- Hallucinations are obvious

✅ Task 2: Claim–Evidence Extraction

Required output format
| Claim | Evidence snippet | Citation (source_id, chunk_id) |

Why this task works:

- Directly tests grounding
- Exposes fake or vague citations
- Translates perfectly into Phase 2 RAG

Phase 1 — Test Cases

You need **2 test cases per task** (4 total).

You don't need a full corpus yet — just **specific papers or reports**.

Task 1: Paper Triage — Test Cases

- **Test Case T1-A:** Empirical study measuring AI coding assistant productivity
- **Test Case T1-B:** Survey or industry report on AI tools in software development

Task 2: Claim–Evidence Extraction — Test Cases

- **Test Case T2-A:** Study focusing on junior developers or students
- **Test Case T2-B:** Study comparing junior vs senior developers

(Next step, I can help you pick *specific papers* if you want.)

Phase 1 — Prompt Kit (Draft)

You need **Prompt A (baseline)** and **Prompt B (improved)** for **each task**.

Below is a **complete, submission-ready draft** for Task 1.
You'll mirror the structure for Task 2.

Task 1 — Paper Triage

🔷 Prompt A (Baseline)

You are given an academic paper about AI tools in software engineering.
Summarize the paper using the following fields:

- "Contribution"
- "Method"
- "Data"
- "Findings"
- "Limitations"

Minimal instruction. No guardrails. This is intentional.

🔷 Prompt B (Improved / Structured)

You are analyzing an academic paper about AI tools in software engineering.

Produce a structured summary using **exactly** the following fields:

1. "Contribution"
2. "Method"
3. "Data"
4. "Findings"
5. "Limitations"

Constraints:

- "Base every claim on the provided text only."
- "If the paper does not clearly state information for a field, write **"Not specified in the paper."**"
- "Do not infer results or speculate beyond the evidence."
- "Be concise and factual."

Output only the five fields listed above.

This prompt adds:

- Structure
- Uncertainty handling
- Anti-hallucination guardrails

Phase 1 — Models

Choose **any 2 accessible models**, for example:

- GPT-4.x
- GPT-3.5
- Claude
- Open-weight model

You will run:

bash

```
2 tasks × 2 test cases × 2 prompts × 2 models = 16 runs
```

Copy code

Phase 1 — Evaluation Sheet (What you score)

Each run gets a row with:

- Task ID
- Test Case ID
- Prompt ID + Model
- Score: Groundedness (1–4)
- Score: Citation correctness / faithfulness (1–4)
- Score: Usefulness (1–4)
- Notes (hallucination, vague claim, overconfidence, etc.)

I can generate a **CSV template** next if you want.

Phase 1 — Analysis Memo (What you'll write later)

You'll answer:

- Which prompt reduced hallucinations?
- Which model overstated confidence?
- Where did models fail most often?

- What constraints worked?
- What design choices will carry into Phase 2 RAG?

Next steps (pick one)

Reply with **one** of these and I'll continue immediately:

1. **"Help me pick specific papers for test cases"**
2. **"Write Prompt A/B for Task 2 (claim–evidence extraction)"**
3. **"Create the evaluation sheet template"**
4. **"Draft the 1–2 page framing brief text"**

You're officially past the hardest part — the rest is execution.

Should I be putting all this in a google doc? What are we supposed to turn in via a document.

Great question — this is exactly the right thing to clarify **before** you start writing stuff in random places.

Short answer (human version)

Yes, **use a Google Doc** for the *written pieces*.
Use a **GitHub repo** for prompts, tables, and anything that looks like "artifacts."

Think of Phase 1 as **half document, half structured files**.

What goes where (very clearly)

✅ Google Doc (or PDF later) — *narrative + reflection*

This is what graders actually *read* end-to-end.

You should put **these three things** in one Google Doc:

1️⃣ Framing Brief (1–2 pages)

This includes:

- Domain
- Main research question
- Sub-questions

- Scope (what's included / excluded)
- Why this domain matters

📌 This reads like a short research proposal.

2️⃣ Analysis Memo (1–2 pages)

This includes:

- What you observed across the 16 runs
- Where models hallucinated or overclaimed
- Prompt A vs Prompt B differences
- Model differences
- What you'll change in Phase 2

📌 This is reflective + analytical.

3️⃣ AI Usage Disclosure (1 page, required)

- Tool name
- What you used it for
- What you edited manually
- Responsibility statement

📌 Often appended at the end.

👉 Result: ONE Google Doc

Later exported as **PDF** and uploaded or linked.

❌ What should *not* live only in a Google Doc

Graders don't want prompts or tables trapped in prose.

These go in **GitHub**.

✅ GitHub Repo — *artifacts + structure*

Put these in your repo:

📁 `prompts/`

- Prompt A and Prompt B (Markdown)

- One file per task
- Include short comments explaining guardrails

📁 `eval/`

- Evaluation sheet (CSV or Markdown table)
- 16 rows (one per run)
- Scores + notes

📁 `report/`

- Optional: a copy of your framing brief & analysis memo as Markdown
- Not required but looks professional

📄 `README.md`

Explain:

- What Phase 1 is
- What domain you chose
- How to interpret the evaluation sheet

What you actually "turn in"

This depends on your class, but **typically**:

Submission includes:

- 🔗 **GitHub repo link**
- 📄 **PDF of Google Doc** (framing + analysis + AI disclosure)
- 🎥 (No demo needed for Phase 1)

If they only allow **one link** → put the Google Doc link **inside the README**.

Suggested Phase 1 workflow (low stress)

Here's the cleanest way to work:

1. **Create Google Doc**
   - Title: *Phase 1 — Personal Research Portal*
   - Sections:
     - Framing Brief
     - Analysis Memo
     - AI Usage Disclosure
2. **Create GitHub repo**
   - Add folders: `prompts/`, `eval/`
   - Drop structured stuff there

3. **Cross-reference**
    - Doc says: “See evaluation table in repo”
    - Repo README links back to the doc

Graders *love* this.

TL;DR (save this)

Item	Where
Framing brief	Google Doc
Analysis memo	Google Doc
AI usage log	Google Doc
Prompts	GitHub
Evaluation table	GitHub
Final submission	PDF + GitHub link

If you want, next I can:

- Give you a **Google Doc outline you can copy**
- Generate a **Phase 1 README.md**
- Create a **folder + file naming convention**
- Help you write the **framing brief text in your voice**

Just tell me which one 👍

Phase 1 deliverables (submit together)

- Framing brief (1–2 pages).
- Prompt kit (Markdown): prompts + why each constraint exists.
- Evaluation sheet (CSV/Google Sheet/Markdown table): scores + notes per run (for MVP, include 16 rows - one per run).
- Analysis memo (1–2 pages): patterns, failures, and Phase 2 design choices.

Perfect — this screenshot actually answers the question *precisely*. Let’s translate it into **what you physically create and submit**, with zero ambiguity.

What you must submit for Phase 1 (exactly)

You are submitting **4 deliverables**, and **they do NOT all live in one place**.

Think of it as:

- **2 narrative documents**
- **2 structured artifacts**

✅ Deliverable 1: Framing brief (1–2 pages)

What this is

A short research framing document:

- Domain
- Main research question
- Sub-questions
- Scope (what's included / excluded)
- Why this domain matters

Where this should live

📄 **Google Doc** (later exported as PDF if needed)

This is prose. Graders read this like a mini research proposal.

✅ Deliverable 2: Prompt kit (Markdown)

> "prompts + why each constraint exists"

What this is

A **Markdown file**, not prose paragraphs.

It includes:

- Prompt A and Prompt B (for each task)
- Clear sections like:
  - Intent
  - Inputs
  - Outputs
  - Constraints / guardrails
  - Why each constraint exists
  - Failure modes to watch for

Where this should live

📁 **GitHub repo**

File example:

```
prompts/
  task1_paper_triage.md
  task2_claim_evidence.md
```

Copy code

❗ Do **not** put this only in a Google Doc — graders expect Markdown.

✅ Deliverable 3: Evaluation sheet (16 rows)

> "CSV / Google Sheet / Markdown table"

What this is

A **table**, one row per run:

- Task
- Test case
- Prompt
- Model
- Scores (1–4)
- Notes / failure tags

Where this should live (best options)

✅ **CSV in GitHub** (recommended)

OR

✅ Google Sheet (linked)

Example:

```bash
eval/
  phase1_eval.csv
```

Copy code

You can *generate it* in Google Sheets, then export to CSV.

✅ Deliverable 4: Analysis memo (1–2 pages)

> "patterns, failures, and Phase 2 design choices"

What this is

Reflective + analytical writing:

- What failed
- What improved with Prompt B
- Model differences
- What you'll carry into Phase 2

Where this should live

📄 **Google Doc**
(often in the same doc as the framing brief, but a new section)

✅ Final submission structure (what graders expect)

You submit two things together:

1️⃣ A single document (PDF or Google Doc link)

Contains:

- Framing brief
- Analysis memo
- AI usage disclosure (usually appended)

2️⃣ A GitHub repo

Contains:

- Prompt kit (Markdown)
- Evaluation sheet (CSV or MD table)
- README explaining what's where

If they only allow **one link**, put the Google Doc link in the README.

TL;DR (copy this)

Deliverable	Format	Where
Framing brief	Prose (1–2 pages)	Google Doc
Prompt kit	Markdown	GitHub
Evaluation sheet	CSV / Sheet / MD table	GitHub
Analysis memo	Prose (1–2 pages)	Google Doc

Next best step (do this now)

1. Create **one Google Doc** titled
   "Phase 1 – Personal Research Portal"
2. Create **one GitHub repo**
3. Add folders:

```bash
prompts/
eval/
```

Copy code

If you want, next I can:

- Give you a **copy-paste Google Doc outline**
- Write a **Prompt Kit Markdown template**
- Generate a **ready-to-fill evaluation CSV**
- Help you draft the **framing brief text in your voice**

Just say the word.

Great! Can you give me the Framing Brief to paste into the google doc

Absolutely — here's a **clean, polished Framing Brief** you can **paste directly into your Google Doc**.
It's written in a clear, student voice (not overly "AI-ish"), hits every rubric requirement, and leaves room for small edits if you want to personalize it.

Phase 1 — Framing Brief

Personal Research Portal (PRP)

Research Domain

The domain of this project is the impact of AI-assisted tools (such as large language models and coding assistants) on software engineering practice. In particular, this research focuses on how these tools influence developer productivity, code quality, and skill development, with an emphasis on early-career and junior software engineers.

As AI tools become increasingly integrated into development workflows, they are changing how developers write, debug, and reason about code. While many claims

suggest significant productivity gains, there is also growing concern about over-reliance, reduced understanding, and long-term effects on developer skills. This domain is well-suited for a research-focused system because it contains a mix of empirical studies, industry reports, and ongoing debates with incomplete or conflicting evidence.

Main Research Question

How do AI tools affect software engineer productivity, code quality, and skill development, particularly for early-career developers?

This question is intentionally framed to capture both positive and negative impacts while remaining grounded in empirical evidence rather than anecdotal claims. It also aligns well with research tasks that require citation-backed answers and explicit handling of uncertainty.

Sub-Questions

To break down the main research question into retrievable and evaluable components, this project explores the following sub-questions:

1. What productivity gains or losses are empirically measured when software engineers use AI-assisted tools?
2. How does AI assistance affect code quality, correctness, and maintainability?
3. How do the impacts of AI tools differ between junior or early-career developers and senior developers?
4. What risks or negative effects (such as over-reliance or skill degradation) are identified in the literature?
5. Where do studies disagree, or where is evidence inconclusive or limited?
6. What long-term impacts on developer skill formation are suggested but not yet empirically validated?

These sub-questions will later inform retrieval queries and synthesis tasks in subsequent phases of the project.

Scope and Boundaries

This project focuses on evidence related to software engineering tasks such as coding, debugging, testing, and code comprehension. Priority is given to empirical studies, controlled experiments, and reputable industry or academic reports that explicitly evaluate the effects of AI tools on developers.

The scope intentionally excludes:

- Purely opinion-based articles or blog posts without supporting data
- General discussions of AI ethics not tied to software engineering practice
- AI applications outside the context of programming or developer workflows

By narrowing the scope, the project aims to maintain a clear evidentiary standard and ensure that claims can be evaluated against concrete sources.

Purpose and Motivation

The purpose of this research is to better understand how AI tools are reshaping the work of software engineers, especially those early in their careers who may be most influenced by these technologies. As AI-assisted development becomes more common in both educational and professional settings, it is important to distinguish between claims that are well-supported by evidence and those that are speculative or overstated.

This framing also directly supports the goals of the Personal Research Portal by emphasizing traceability, citation correctness, and transparency around uncertainty. The insights gained in Phase 1 will inform the design of a research-grade retrieval and grounding system in Phase 2, and ultimately a usable research portal in Phase 3.

If you want, next I can:

- Draft the **Analysis Memo** in the same voice
- Create the **Prompt Kit Markdown files**
- Generate a **ready-to-fill evaluation CSV**
- Or help you lightly **personalize this framing** so it sounds even more like you

Great now that we have that, whats the next step?

Awesome — with the **Framing Brief locked**, you're officially past the "what is this project?" hurdle.
From here on out, Phase 1 becomes **mechanical and very doable**.

The next step (do this next, not everything at once)

🔑 Step 2: Create your Prompt Kit

This is the *core* of Phase 1.

Why this comes next:

- Prompts drive the experiments
- Everything else (runs, evaluation, analysis) depends on them
- It's easier to write prompts *before* collecting results

What the Prompt Kit actually is

You need:

- **2 research tasks**
- **2 prompt variants per task**
  - Prompt A = baseline
  - Prompt B = improved / guarded

That's **4 prompts total**, written in **Markdown**.

Your already-chosen tasks (recap)

We locked these earlier because they're ideal for your domain:

✅ Task 1: Paper Triage

Output: Contribution | Method | Data | Findings | Limitations

✅ Task 2: Claim–Evidence Extraction

Output: Claim | Evidence snippet | Citation (source_id, chunk_id)

What you should do *right now*

1️⃣ Create this file structure in your GitHub repo

```
prompts/
  paper_triage.md
  claim_evidence_extraction.md
```

Copy code

2️⃣ Inside each file, you will include:

For **each task**:

- Prompt A (baseline)
- Prompt B (improved)
- A short explanation of **why each constraint exists**

The assignment *explicitly* asks for this.

What I recommend we do next (together)

👉 **Next best step:**
I write the full Prompt Kit for you (both tasks, both prompts), ready to paste into Markdown.

That gives you:

- A concrete artifact in your repo
- Something you can immediately start running experiments with
- Zero ambiguity about what graders expect

After the Prompt Kit (roadmap so you're not anxious)

Once prompts are done, the remaining Phase 1 steps are:

1. **Pick 4 concrete test cases** (specific papers/reports)
2. **Run the 16 experiments**
3. **Fill out the evaluation sheet**
4. **Write the analysis memo**

Each step depends on the previous one — no surprises.

Your move

Reply with **one** of these and I'll continue immediately:

- **"Write the full Prompt Kit (both tasks)"** ← recommended
- "Help me choose specific papers for test cases"
- "Create the evaluation sheet template"
- "Outline the analysis memo"

You're in the smooth part of Phase 1 now.

Write the full Prompt Kit (both tasks)

Perfect — below is a **complete, submission-ready Prompt Kit** for **both tasks**, written exactly the way graders expect it.

You can **paste this directly into Markdown files** in your GitHub repo.

Prompt Kit — Phase 1

Domain: AI tools and software engineering productivity
Purpose: Evaluate how different prompt structures affect grounding, faithfulness, and uncertainty handling in research tasks.

📄 File 1: `prompts/paper_triage.md`

markdown

```markdown
# Task: Paper Triage

## Intent
Summarize a research paper on AI-assisted software engineering in a struct

## Inputs
- A single academic paper or report (PDF text or excerpt) related to AI to

## Required Output Structure
The output must contain exactly the following five fields:
1. Contribution
2. Method
3. Data
4. Findings
5. Limitations

---

## Prompt A — Baseline

You are given an academic paper about AI tools in software engineering.

Summarize the paper using the following fields:
- Contribution
- Method
- Data
- Findings
```

```
- Limitations

---

## Prompt B — Structured with Guardrails

You are analyzing an academic paper about AI-assisted tools in software en

Produce a structured summary using **exactly** the following five fields:
1. Contribution
2. Method
3. Data
4. Findings
5. Limitations

### Constraints / Guardrails
- Base all statements strictly on the content of the provided paper.
- Do not infer or speculate beyond what is explicitly supported by the tex
- If the paper does not clearly specify information for a field, write: **
- Be concise, factual, and neutral in tone.
- Do not introduce external knowledge or assumptions.

Output only the five fields listed above.

---

## Why These Constraints Exist
- **Exact fields:** Ensures outputs are comparable across runs and models.
- **"Not specified" requirement:** Prevents hallucination when information
- **No speculation:** Tests whether the model can resist filling gaps with
- **Concise, neutral tone:** Encourages research-style summarization rathe

## Failure Modes to Watch For
- Hallucinated methods or datasets.
- Overstated findings not supported by the paper.
- Vague or generic limitations not grounded in the text.
- Ignoring the required structure.
```

Copy code

File 2: `prompts/claim_evidence_extraction.md`

markdown

```
# Task: Claim–Evidence Extraction

## Intent
```

```
Extract concrete claims from research papers and pair them with direct sup

## Inputs
- A research paper or excerpt related to AI tools and software engineering
- Each paper is associated with a known `source_id`.

## Required Output Structure
A table with exactly five rows and the following columns:
| Claim | Evidence Snippet | Citation (source_id, chunk_id) |

---

## Prompt A — Baseline

From the provided paper, extract five key claims about the impact of AI to

For each claim, include:
- The claim
- A supporting quote or snippet from the text
- A citation

Present your answer in a table.

---

## Prompt B — Structured with Guardrails

You are extracting claims and evidence from a research paper about AI-assi

Produce a table with **exactly five rows** and the following columns:
| Claim | Evidence Snippet | Citation (source_id, chunk_id) |

### Constraints / Guardrails
- Each claim must be directly supported by the provided evidence snippet.
- Evidence snippets must be copied verbatim or closely paraphrased from th
- Citations must use the format: **(source_id, chunk_id)**.
- Do not invent claims, evidence, or citations.
- If the paper does not contain enough information to extract five well-su
- Do not rely on external knowledge.

---

## Why These Constraints Exist
- **Fixed table format:** Enables consistent evaluation across models and
- **Direct evidence requirement:** Forces grounding and discourages abstra
- **Explicit citation format:** Tests whether models can correctly attribu
- **Insufficient evidence rule:** Rewards uncertainty handling rather than
```

```
## Failure Modes to Watch For
- Claims that are broader than the evidence supports.
- Fabricated or vague citations.
- Evidence snippets that do not actually support the stated claim.
- Reworded "common knowledge" instead of paper-specific findings.
```

Copy code

✅ What you've now completed

With this Prompt Kit, you now have:

- ✔ Both required research tasks
- ✔ Prompt A and Prompt B for each task
- ✔ Clear justification for constraints
- ✔ Explicit failure modes (graders love this)

This fully satisfies the **"Prompt kit (Markdown)"** deliverable.

➡ SOON Next step (recommended)

The clean next move is to **choose your 4 concrete test cases** (specific papers/reports), because:

- Prompts are done
- Test cases determine everything else (runs, eval, analysis)

If you want, next I can:

- Help you **pick specific papers** (very fast)
- Create the **evaluation sheet (CSV)** ready to fill
- Outline the **analysis memo** so you know what to look for while running experiments

Just tell me what you want to do next.

great now lets look for some specific papers

Awesome — this is the **right next step**. We'll pick **specific, credible papers** that:

- Clearly study **AI coding tools**
- Make **testable claims** (productivity, quality, juniors vs seniors)
- Are easy to use for **Paper Triage** and **Claim–Evidence Extraction**
- Give you **contrast** (empirical vs survey, juniors vs mixed)

You only need **4 papers for Phase 1** (not the full Phase 2 corpus yet).

Phase 1 Test Case Papers (Recommended Set)

I'll label these so you can drop them straight into your evaluation sheet.

🧪 Task 1: Paper Triage — 2 Test Cases

T1-A: Controlled empirical study (productivity + quality)

Paper:
"The Impact of AI-Assisted Code Generation on Developer Productivity"
(aka the GitHub Copilot controlled experiment paper)

Why this paper is excellent

- Randomized controlled study
- Measures **task completion time** and **success rate**
- Very concrete methods + data
- Clear limitations section (short-term, task-specific)

What it tests

- Can the model accurately extract *methods vs findings*?
- Does it hallucinate long-term conclusions?

📌 Perfect for spotting **overgeneralization failures**.

T1-B: Industry / survey-style report

Paper / Report:
"The State of AI in Software Development" (GitHub / Microsoft / industry survey)

Why this works

- Aggregated survey data
- Self-reported productivity and usage
- Less rigorous than T1-A (on purpose)

What it tests

- Can the model distinguish *perceived productivity* from *measured productivity*?
- Does it correctly note methodological limitations?

📌 Great for testing **critical reading vs hype repetition**.

🔍 Task 2: Claim–Evidence Extraction — 2 Test Cases

T2-A: Junior / student-focused study

Paper:
"Do Programming Assistants Improve Learning Outcomes for Novice Programmers?"
(or similar study focusing on students / novices using AI or autocomplete tools)

Why this is strong

- Focuses on **early-career / novice developers**
- Often shows **mixed or conditional effects**
- Forces careful claim extraction

What it tests

- Does the model correctly limit claims to novices?
- Can it extract nuanced risks (dependency, shallow learning)?

📌 Ideal for catching **claim scope errors**.

T2-B: Comparative or mixed-experience study

Paper:
"How AI Coding Tools Affect Developers With Different Levels of Experience"
(or studies comparing junior vs senior outcomes)

Why this works

- Explicit comparison between experience levels
- Often reports **uneven benefits**
- Naturally supports disagreement or nuance

What it tests

- Can the model correctly attribute claims to subgroups?
- Does it collapse differences into a single claim?

📌 Excellent for **citation precision testing**.

What you should do *next* (very concrete)

Step 1: Lock these as your Phase 1 test cases

In your notes or evaluation sheet, label them exactly like this:

- **T1-A:** Copilot controlled experiment (empirical)

- **T1-B:** Industry survey report
- **T2-A:** Novice / student-focused study
- **T2-B:** Junior vs senior comparison study

You don't need perfect titles yet — consistency matters more.

Step 2: Get PDFs or excerpts

For Phase 1, it's totally acceptable to:

- Download PDFs
- Or copy **relevant sections** (methods + results)
- Or use official report PDFs

You are **not** building the manifest yet.

Step 3: Decide your models

Pick your two models now (e.g., GPT-4.x and GPT-3.5), so runs are consistent.

After this (what's coming next)

Once papers are locked:

1. Run **16 prompt executions**
2. Fill out the **evaluation CSV**
3. Write the **analysis memo** using observed failures

Want me to go one level deeper?

I can:

- 🔗 Give you **exact paper titles + links**
- 🧠 Suggest **which sections to feed the model**
- 📄 Create the **evaluation CSV pre-filled with IDs**
- ✍️ Draft a **scoring guide** so grading is easy

Just tell me which one you want next.

can you get the links for those

Here are **actual links to research papers** you can use as **Phase 1 test cases** — reliable, citable, and directly relevant to your topic on AI impacts in software engineering. I'll break them down by task:

📄 Task 1 – Paper Triage (2 Papers)

1. "The Impact of AI on Developer Productivity: Evidence from GitHub Copilot"

This is a controlled experiment that measures how productivity changes when developers use an AI coding assistant. arXiv

📌 **Link (arXiv preprint):**
https://arxiv.org/abs/2302.06590
(it also has a PDF you can download) arXiv

This is perfect for **Paper Triage** because:

- It has clear methodology
- Productivity outcome measures
- Results you can evaluate for grounding and citations

2. "How AI Impacts Skill Formation" (2026, preprint)

This paper investigates how AI assistance affects developers' skill acquisition, focusing on novices and conceptual understanding, not just speed. arXiv

📌 **Link (arXiv preprint):**
https://arxiv.org/abs/2601.20245

This one is great for showing a *nuanced effect* rather than just productivity numbers — ideal for comparing models' nuance. arXiv

📋 Task 2 – Claim–Evidence Extraction (2 Papers)

3. "The Good and Bad of AI Tools in Novice Programming Education"

This examines how novice programmers use AI and the benefits/drawbacks, with implications for learning outcomes. MDPI

📌 **Link (MDPI journal article):**
https://www.mdpi.com/2227-7102/14/10/1089

Use this to extract claims about AI impacts on student learning, over-reliance, and conceptual understanding. MDPI

4. *Alternative (or second choice)* — ACM or similar study on AI and productivity

Optionally, instead of or alongside the novice education paper, you could use this ACM article on Copilot's impact on productivity. Communications ...

📌 **Link (ACM article):**
https://cacm.acm.org/research/measuring-github-copilots-impact-on-productivity/

This one works for claim–evidence extraction on mixed evidence about productivity and experience level differences. Communications ...

Quick Summary — Ready to Insert into Spiral/Sheet

You can use these as your **four test cases**:

ID	Title	Link
T1-A	The Impact of AI on Developer Productivity: Evidence from GitHub Copilot	https://arxiv.org/abs/2302.06590 arXiv
T1-B	How AI Impacts Skill Formation	https://arxiv.org/abs/2601.20245 arXiv
T2-A	The Good and Bad of AI Tools in Novice Programming Education	https://www.mdpi.com/2227-7102/14/10/1089 MDPI
T2-B	Measuring GitHub Copilot's Impact on Productivity (ACM)	https://cacm.acm.org/research/measuring-github-copilots-impact-on-productivity/ Communications ...

Usage tips

How to feed these into your prompts:

- For **Paper Triage**, include the *Methods*, *Results*, and *Limitations* sections from the PDF or HTML.

- For **Claim–Evidence Extraction**, focus on *Results* + *Discussion* where claims are supported by specific data.

Optional Additional Sources (not required, but useful)

If you want **backup evidence** for analysis or Phase 2 later:

- Stack Overflow 2025 AI tools survey (developer adoption and perceptions) — https://survey.stackoverflow.co/2025/ai survey.stackoverf... (good for contextual claims)

If you want, I can next help you with:

✅ deciding which **text chunks** to extract and give to models,
✅ building a **spreadsheet template** with these test cases pre-filled,
or
✅ outlining your **evaluation rubric** for Phase 1.

Just tell me which you want!

Sources

where should i add this table? on a google sheet?

Yes — **a Google Sheet is the right place for this table**, and you'll later export it as a CSV for your GitHub repo.

Here's the **clean, no-confusion workflow**.

Where this table should live

✅ Primary workspace: Google Sheet

Use a Google Sheet while you're actively working because:

- Easy to enter scores and notes
- Easy to filter/sort by model, prompt, task
- Easy to collaborate or double-check yourself
- Easy to export to CSV later (required)

📌 Think of the Sheet as your *working evaluation log*.

Where it ends up (for submission)

Stage	Location
Working	Google Sheet
Final artifact	CSV in GitHub repo (`eval/phase1_eval.csv`)
Optional	Link to Google Sheet in README

Graders usually prefer **CSV in the repo**, but they don't mind if it *originated* in Sheets.

How to structure the Google Sheet (do this exactly)

Sheet name

```
Phase 1 Evaluation
```

Columns (copy/paste this header row)

```scss
Run_ID
Task_ID
Test_Case_ID
Paper_Title
Prompt_ID
Model
Groundedness_Score (1–4)
Citation_Correctness_Score (1–4)
Usefulness_Score (1–4)
Failure_Tags
Notes
```

Copy code

This directly maps to the rubric.

How many rows should you have?

You will end with **16 rows**:

```mathematica
2 Tasks
× 2 Test Cases
```

```
× 2 Prompts
× 2 Models
= 16 runs
```

Copy code

One row = **one model output**.

How to label things (important for clarity)

Task_ID

- `T1` → Paper Triage
- `T2` → Claim–Evidence Extraction

Test_Case_ID

- `T1-A`, `T1-B`
- `T2-A`, `T2-B`

Prompt_ID

- `A` (Baseline)
- `B` (Structured)

Model

- `GPT-4.x`
- `GPT-3.5`

  (or whatever you use — just be consistent)

Where the paper table you asked about fits

The **paper list table** (titles + links) is **not the evaluation sheet**.

You have two good options:

Option 1 (recommended): Add a second tab

In the **same Google Sheet**, add another tab called:

```
Phase 1 Test Cases
```

Columns:

```arduino
Test_Case_ID
Task
Paper_Title
```

```
Authors
Year
Link
Notes
```

Copy code

This keeps everything organized in one place.

Option 2: Put it in the Google Doc

You *can* put the paper list table in the **Framing Brief** appendix, but:

- It's harder to reuse
- Harder to export later

I'd only do this if your instructor explicitly wants it in prose.

My strong recommendation (do this)

1. Create **one Google Sheet**
2. Add **two tabs**:
   - `Phase 1 Test Cases`
   - `Phase 1 Evaluation`
3. Work entirely in Sheets
4. Export `Phase 1 Evaluation` → CSV → GitHub

This matches how the project is graded.

Next step after this

Once the Sheet exists, the next concrete step is:

➡️ **Run the first paper through Prompt A + Prompt B**
➡️ Paste outputs into your notes
➡️ Score them in the sheet

If you want, next I can:

- Pre-fill the Sheet rows for you
- Help you score the **first run together**
- Show you **what a "4 vs 2 vs 1" looks like** in practice

give an example of how the sheets shoudl look

Perfect — here's a **concrete example** of exactly how your Google Sheet should look. I'll show you **both tabs** with **realistic example rows**, so you can copy this almost verbatim.

✅ Google Sheet Structure (Phase 1)

You should have **ONE Google Sheet** with **TWO tabs**.

🗂️ Tab 1: `Phase 1 Test Cases`

This tab just documents **what papers you're using**.
You fill this out **once**.

Header row (copy this exactly)

arduino

```
Test_Case_ID | Task | Paper_Title | Authors | Year | Link | Notes
```

Copy code

Example rows

csharp

```
T1-A | Paper Triage | The Impact of AI on Developer Productivity: Evidence
T1-B | Paper Triage | Measuring GitHub Copilot's Impact on Productivity |
T2-A | Claim-Evidence Extraction | The Good and Bad of AI Tools in Novice
T2-B | Claim-Evidence Extraction | How AI Coding Tools Affect Developers W
```

Copy code

📌 This tab is **reference only** — no scoring here.

📊 Tab 2: `Phase 1 Evaluation`

This is the **main graded artifact**.
You will end with **16 rows** here.

Header row (copy this exactly)

nginx

```
Run_ID | Task_ID | Test_Case_ID | Paper_Title | Prompt_ID | Model | Ground
```

Copy code

Example rows (realistic, filled out)

csharp

```csharp
1 | T1 | T1-A | Impact of AI on Developer Productivity | A | GPT-4 | 3 | 3
2 | T1 | T1-A | Impact of AI on Developer Productivity | B | GPT-4 | 4 | 4
3 | T1 | T1-A | Impact of AI on Developer Productivity | A | GPT-3.5 | 2 |
4 | T1 | T1-A | Impact of AI on Developer Productivity | B | GPT-3.5 | 3 |
```

Copy code

⬆️ These four rows represent:

- Same paper
- Same task
- Two prompts
- Two models

You'll repeat this pattern until you have **16 rows total**.

📁 Common values for `Failure_Tags`

Use short, consistent tags (comma-separated if needed):

- `Hallucination`
- `Overgeneralization`
- `Vague citation`
- `Wrong scope`
- `Missing limitation`
- `Overconfident`
- `Structure violation`
- `None`

Graders like consistency here.

🎯 What a "good" row looks like to graders

A strong row:

- Has **specific notes**, not "good/bad"
- Explains *why* you gave a 2 vs 4
- Ties directly to **grounding or evidence**

Bad note ❌

> "Seems wrong"

Good note ✅

> "Claims productivity improvements beyond the measured task duration and sample size."

🔄 Workflow tip (important)

Do **NOT** paste full model outputs into the sheet.
Instead:

- Keep outputs in a scratch doc or notes
- Score + summarize in the Sheet

The Sheet is for **evaluation**, not storage.

✅ You're doing this exactly right

Once this Sheet exists, the next step is simply:
➡️ Run **T1-A** through **Prompt A** on **Model 1**

If you want, next I can:

- Walk through **scoring a real output together**
- Pre-fill all 16 rows (structure only)
- Help you write **strong failure notes quickly**

Just tell me 👍

can i get it in csv format for easy copy and paste

Absolutely — here are **clean CSV templates** you can **copy-paste directly** into Google Sheets (or Excel).
I'll give you **both tabs** as separate CSV blocks.

✅ CSV 1: Phase 1 Test Cases

Create a tab called `Phase 1 Test Cases`, then paste this:

```csv
Test_Case_ID,Task,Paper_Title,Authors,Year,Link,Notes
T1-A,Paper Triage,The Impact of AI on Developer Productivity: Evidence fro
T1-B,Paper Triage,Measuring GitHub Copilot’s Impact on Productivity,GitHub
T2-A,Claim–Evidence Extraction,The Good and Bad of AI Tools in Novice Prog
T2-B,Claim–Evidence Extraction,How AI Coding Tools Affect Developers With
```

Copy code

📌 You can fill in the missing link later for `T2-B` if you swap papers.

✅ CSV 2: Phase 1 Evaluation Sheet

Create a second tab called `Phase 1 Evaluation`, then paste this:

```csv
Run_ID,Task_ID,Test_Case_ID,Paper_Title,Prompt_ID,Model,Groundedness_Score
1,T1,T1-A,Impact of AI on Developer Productivity,A,GPT-4,,,,"",
2,T1,T1-A,Impact of AI on Developer Productivity,B,GPT-4,,,,"",
3,T1,T1-A,Impact of AI on Developer Productivity,A,GPT-3.5,,,,"",
4,T1,T1-A,Impact of AI on Developer Productivity,B,GPT-3.5,,,,"",
5,T1,T1-B,Measuring GitHub Copilot’s Impact on Productivity,A,GPT-4,,,,"",
6,T1,T1-B,Measuring GitHub Copilot’s Impact on Productivity,B,GPT-4,,,,"",
7,T1,T1-B,Measuring GitHub Copilot’s Impact on Productivity,A,GPT-3.5,,,,"
8,T1,T1-B,Measuring GitHub Copilot’s Impact on Productivity,B,GPT-3.5,,,,"
9,T2,T2-A,Novice Programming AI Tools,A,GPT-4,,,,"",
10,T2,T2-A,Novice Programming AI Tools,B,GPT-4,,,,"",
11,T2,T2-A,Novice Programming AI Tools,A,GPT-3.5,,,,"",
12,T2,T2-A,Novice Programming AI Tools,B,GPT-3.5,,,,"",
13,T2,T2-B,Experience Level AI Coding Tools,A,GPT-4,,,,"",
14,T2,T2-B,Experience Level AI Coding Tools,B,GPT-4,,,,"",
15,T2,T2-B,Experience Level AI Coding Tools,A,GPT-3.5,,,,"",
16,T2,T2-B,Experience Level AI Coding Tools,B,GPT-3.5,,,,""
```

Copy code

📌 This pre-fills **all 16 required runs** — you just add scores and notes.

🔁 How to use this (important)

1. Paste CSV → Google Sheets
2. Run experiments
3. Fill in:
   - Scores (1–4)

- Failure tags
- Notes

4. When done:
   - **File → Download → CSV**
   - Upload to GitHub as `eval/phase1_eval.csv`

🧠 Pro tip (graders love this)

In **Notes**, always mention *why*:

- "Invented long-term claims"
- "Correctly states limitation"
- "Evidence snippet does not support claim"

Specific > generic.

If you want, next I can:

- Help you **run + score your first row**
- Show examples of **what earns a 4 vs a 2**
- Help you **swap or lock the T2-B paper**

You're very much on track 🚀

it doesnt look like a table, it just appears all in one cell for each row

Perfect screenshot — this is a **very common CSV → Sheets gotcha**, and you didn't do anything wrong 👍
Google Sheets just didn't auto-split the commas.

Let's fix it in **30 seconds**.

Why this happened (quick explanation)

When you paste CSV **directly into a cell**, Sheets treats it as plain text, not as structured data.

You just need to tell Sheets:

> "Hey, split this text by commas."

✅ Fix option 1 (fastest – do this now)

Step-by-step in Google Sheets

1. **Select column A** (click the "A" at the top)
2. Go to **Data → Split text to columns**
3. In the small dropdown that appears:
   - Choose **Comma**

💥 Instantly becomes a proper table.

✅ Fix option 2 (best practice for the future)

Instead of pasting:

1. **File → Import**
2. Upload the CSV file
3. Choose:
   - Insert new sheet(s)

This avoids the issue entirely next time.

What it should look like after splitting

You should now see:

A	B	C	D	E	F	G
Test_Case_ID	Task	Paper_Title	Authors	Year	Link	N

A	B	C	D	E	F	G
T1-A	Paper Triage	The Impact of AI on Developer Productivity…	Noy et al.	2023	https://arxiv.org/…	Co ex

If you **don't see column headers separated**, redo the split.

One small cleanup suggestion (optional but nice)

After splitting:

- **Freeze row 1**
  View → Freeze → 1 row
- **Wrap text** in the Notes column
  Format → Wrapping → Wrap

Makes grading easier.

You're exactly on track

Once this table looks right:

- Leave this tab alone (it's done)
- Switch to the **Phase 1 Evaluation** tab next
- Start running **T1-A → Prompt A → Model 1**

If you want, next I can:

- Walk you through **your first model run**
- Help you decide **what section of the paper to paste**
- Help you score your **first row correctly**

Just say what you want to do next 👍

Test_Case_ID	Task	Paper_Title	Authors	Year	Link	Notes
T1-A	Paper Triage	The Impact of AI on Develope	Various	2023	https://arxiv.org/abs	Controlled experiment measuring task completion time and success rate
T1-A	Paper Triage	How AI Impacts Skill Formati	Various	2026	https://arxiv.org/abs	How AI assistance affects developers' skill acquisition and conceptual understanding
T2-A	Claim–Evidence Extraction	The Good and Bad of AI Tools	Various	2024	https://www.mdpi.cc	Focuses on novice programmers and learning outcomes
T2-B	Claim–Evidence Extraction	Measuring GitHub Copilot's In	GitHub / Microsoft	2022	https://cacm.acm.or	Industry-style report based on observational and survey data

Nice — this **looks like a proper table now** 👍

You're basically there. Just a couple of **small but important fixes** before you move on.

✅ What's good (no changes needed)

- Columns are correct ✅
- Tasks are clearly labeled ✅
- Links are present and clickable ✅
- Notes are concise and informative ✅
- This is **exactly** the kind of table graders expect

⚠️ Small fixes to make (important)

1️⃣ Duplicate Test Case ID

You currently have **two** `T1-A` **rows**.

You want:

- `T1-A`
- `T1-B`

Fix:

Change the second row's `Test_Case_ID` from:

```
T1-A → T1-B
```

Copy code

2️⃣ Task alignment (minor but worth fixing)

Right now you have:

- **T1-A** → Paper Triage ✅
- **T1-B** → Paper Triage ✅
- **T2-A** → Claim–Evidence Extraction ✅
- **T2-B** → Claim–Evidence Extraction ✅

That's perfect — just double-check you **keep this mapping consistent** when you run experiments.

3️⃣ Optional polish (not required, but nice)

Authors

If you want to look extra clean:

- Replace `Various` with:

- `Noy et al.` (for Copilot paper)
- `GitHub / Microsoft` (for CACM report)

Not required, but it improves clarity.

4️⃣ Sanity check on the 2026 paper (important question)

You listed a **2026 arXiv paper** ("How AI Impacts Skill Formation").

Quick check:

- If this is a **real arXiv preprint you accessed**, you're fine.
- If you're unsure, you can swap it with a safer option.

If you want a safer alternative for **T1-B**, say the word and I'll suggest one.

✅ After these fixes, this tab is DONE

You will **not touch this again** in Phase 1.

➡️ What to do next (very concrete)

Next step: Start running experiments

Do this in order:

1. Open **T1-A** paper
2. Copy:
   - Abstract
   - Methods
   - Results
   - Limitations
3. Run:
   - **Paper Triage → Prompt A → Model 1**
4. Save output somewhere (doc or notes)
5. Go to **Phase 1 Evaluation** tab
6. Fill in **Row 1**

That's it.

If you want help with the very first run

I can:

- Tell you **exactly which sections to paste**
- Help you score **Row 1 together**
- Show you **what earns a 4 vs a 2**

Just say:

> "Let's do the first run together"

Now that I have the papers, what should I do now?

Great question — this is the **exact right moment to ask it**.
You've done all the setup work. From here on, Phase 1 is **execution + observation**.

I'll give you a **very concrete checklist**, then zoom into **what to do right now**.

Big picture: where you are in Phase 1

You now have:

- ✅ Framing brief
- ✅ Prompt kit (both tasks)
- ✅ Test case papers
- ✅ Test case table set up correctly

What remains:

- Run **16 prompt executions**
- Fill the **evaluation sheet**
- Write the **analysis memo**

What you should do next (do this *now*)

🔷 Step 1: Run your first experiment

Don't think about all 16 yet. Just do **one**.

Start with:

- **Task:** Paper Triage
- **Test Case:** T1-A (Copilot empirical study)
- **Prompt:** A (baseline)
- **Model:** Model 1 (e.g., GPT-4)

🔷 Step 2: What to paste into the model (important)

For Phase 1, you do **not** need the full PDF.

Paste:

- Abstract
- Methods
- Results
- Limitations / Discussion

This is enough to test hallucinations and structure.

🔷 Step 3: Save the output (don't lose it)

Do **not** paste full outputs into your evaluation sheet.

Instead:

- Keep a scratch doc or notes file
- Label it clearly, e.g.

  `T1-A | Paper Triage | Prompt A | GPT-4`

🔷 Step 4: Score that run immediately

Go to your **Phase 1 Evaluation** tab and fill **one row**:

- Groundedness (1–4)
- Citation correctness / faithfulness (1–4)
- Usefulness (1–4)
- Failure tags
- Notes (1–2 sentences)

Scoring right away keeps things honest.

How to think while scoring (this matters)

Ask yourself:

- Did it **invent** anything?
- Did it **overgeneralize** beyond the study?
- Did it **miss limitations**?
- Did it handle uncertainty appropriately?

You're not grading the paper — you're grading the **model behavior**.

Step 5: Repeat in a tight loop

Once the first run is done, repeat:

1. Same paper
2. **Prompt B**
3. Same model
4. Score
5. Switch model
6. Repeat

Finish **all 4 runs for T1-A** before moving on.

This makes comparison easier.

Suggested execution order (to stay sane)

Do them in this order:

1. **T1-A** — Paper Triage (4 runs)
2. **T1-B** — Paper Triage (4 runs)
3. **T2-A** — Claim–Evidence Extraction (4 runs)
4. **T2-B** — Claim–Evidence Extraction (4 runs)

When you're "halfway done"

After **8 runs**, pause and:

- Skim your notes
- Notice patterns
- Flag recurring failure modes

This will make your analysis memo much easier.

What *not* to do

🚫 Don't tweak prompts mid-phase
🚫 Don't change papers
🚫 Don't re-score earlier runs retroactively
🚫 Don't wait to score everything at the end

If you want help right now

We can:

- Do **Run #1 together** step-by-step
- Practice scoring a real output
- Pre-write a **notes template** so scoring is fast
- Start drafting the **analysis memo outline** while patterns are fresh

Just say:

> **"Let's do the first run together."**

You're officially in the "just execute" phase — nice work.

So how do I put the prompts into these llms? do i just do copy and paste the simple prompt first and provide the pdf of the paper and then do the same for the complex prompt for every llm on every paper?

Yes — **conceptually you're exactly right**, and I'll refine it into a **clean, repeatable procedure** so you don't accidentally mix things up or bias your results.

Think of Phase 1 as **a controlled experiment**, not casual chatting.

The correct mental model

You are simulating a **research workflow**, not uploading files to a system.

For Phase 1:

- You **manually provide the paper content**
- You **manually paste prompts**
- You **keep everything constant except one variable**

The variables you change are:

- Prompt (A vs B)
- Model (LLM 1 vs LLM 2)

Everything else stays fixed.

Exactly how to run ONE experiment (step by step)

I'll show you the **gold-standard process**. Do this every time.

🔷 Step 1: Prepare the paper text (once per paper)

Open the PDF and copy **only these sections**:

- Abstract
- Methods
- Results
- Limitations / Discussion

Paste them into a scratch doc so you don't re-copy each time.

📌 Do *not* summarize or edit the text.

🔷 Step 2: Open the LLM interface

Examples:

- ChatGPT
- Claude
- Another model UI

Start a **new chat/session** for each run if possible.

Why:

- Prevents context leakage
- Keeps runs independent

🔷 Step 3: Paste Prompt A first

Paste **only** the Prompt A text from your Prompt Kit.

Example (Paper Triage, Prompt A):

> You are given an academic paper about AI tools in software engineering.
> Summarize the paper using the following fields…

Do **not** add commentary.

🔷 Step 4: Immediately paste the paper text

Below the prompt, add:

```sql
--- BEGIN PAPER TEXT ---
[paste abstract + methods + results + limitations]
--- END PAPER TEXT ---
```

Copy code

This is optional but highly recommended — it makes boundaries clear.

🔷 Step 5: Run the model and save the output

Let the model respond.

Then:

- Copy the output into your scratch doc
- Label it clearly, e.g.:

```less
T1-A | Paper Triage | Prompt A | GPT-4
```

Copy code

Do **not** edit the output.

🔷 Step 6: Score the run immediately

Go to your **Phase 1 Evaluation Sheet** and fill **one row**:

- Groundedness
- Citation correctness
- Usefulness
- Failure tags
- Notes

Do this while the output is fresh.

Then what?

🔁 Repeat with controlled changes

For the *same paper*:

1. Prompt B → same model
2. Prompt A → model 2

3. Prompt B → model 2

That's your 4 runs for one test case.

Only then move to the next paper.

What you should NOT do

🚫 Do not rephrase prompts
🚫 Do not summarize the paper before pasting
🚫 Do not upload PDFs and rely on auto-parsing
🚫 Do not reuse the same chat thread
🚫 Do not mix Prompt A and Prompt B in one run

Common student concern (you're not alone)

> "This feels manual and repetitive."

Yes — **that's intentional**.

Phase 1 is about:

- Observing failures
- Seeing where structure helps
- Understanding limits before automating anything

Phase 2 is where automation happens.

Pro tip (saves time)

Create a **template block** in your scratch doc:

yaml

```yaml
RUN LABEL:
Prompt:
Model:
Paper:
Output:
```

Copy code

Paste the model output under "Output" each time.

If you want

Next we can:

- Walk through **one full run together**
- Practice scoring an output
- Decide which two models to use
- Set a **time-efficient schedule** for finishing all 16 runs

Just tell me what you want to do next 👍

2302.06590.pdf
PDF

so in this paper: The Impact of AI on Developer Productivity: Evidence from GitHub Copilot, would i just copy and paste the abstract, study design, and results (without the table), and discussion content?

Yes — **you've got the right idea**, and I'll lock it in very clearly so you don't second-guess yourself.

For **Phase 1**, you should **not** paste the entire PDF. You should paste a **carefully selected subset** that gives the model enough evidence *without noise*.

Using **"The Impact of AI on Developer Productivity: Evidence from GitHub Copilot"**, here's the **exact recommendation**.

✅ What you SHOULD paste for this paper (Paper Triage task)

Paste **only** these sections:

1️⃣ Abstract

This establishes:

- Research question
- Core result (55.8% faster completion)
- High-level contribution

→ From the paper's abstract on page 1 2302.06590

2️⃣ Study Design (Methods)

This is critical because:

- It defines **what was actually measured**
- It prevents hallucinations about causality, scope, or realism

Paste:

- Participant recruitment
- Randomization (control vs treatment)
- Task description (HTTP server)
- Metrics (task success, time to completion)

→ Pages 3–4 ("Study Design") 2302.06590

3️⃣ Results (TEXT ONLY)

Yes:

- Copy the **written explanation** of results

No:

- ❌ Do **not** copy tables or figures
- ❌ Do **not** paste charts

Paste:

- Reported 55.8% speed improvement
- Confidence interval
- Notes on heterogeneity (experience, age, hours)

→ Pages 5–7 ("Results") 2302.06590

4️⃣ Discussion / Limitations

This is *essential* for Phase 1 scoring.

Paste:

- Generalizability limits
- Task standardization caveats
- No measurement of code quality
- Speculation vs evidence boundaries

→ Pages 7–8 ("Discussion") 2302.06590

❌ What you should NOT paste

- Figures (graphs, histograms)

- Tables (heterogeneous effects table)
- References section
- Appendices
- Upwork screenshots
- Survey screenshots

These add noise and don't help the prompt tasks.

✅ Why this is the *correct* approach (for the assignment)

Your instructors want to see:

- Whether the model **overclaims beyond the text**
- Whether structured prompts help the model **respect limitations**
- Whether different models handle **empirical nuance differently**

If you paste everything, you:

- Reduce failure opportunities
- Make Prompt A and Prompt B look artificially similar

So your instinct to exclude tables and visuals is **exactly right**.

🔁 How this fits into your workflow

For **this paper**:

- Use the **same pasted text** for:
  - Prompt A + Model 1
  - Prompt B + Model 1
  - Prompt A + Model 2
  - Prompt B + Model 2

That consistency is crucial.

🧠 Pro tip (use this exact wrapper every time)

When pasting into the LLM, do:

```
--- BEGIN PAPER TEXT ---
[Abstract]

[Study Design]
```

```
[Results — text only]

[Discussion / Limitations]
--- END PAPER TEXT ---
```

Copy code

This makes your runs cleaner and easier to explain in your analysis memo.

You're doing this *exactly* right

Next logical step:

👉 Run **Paper Triage – Prompt A – Model 1** using this pasted text.

If you want, we can:

- Do that **first run together**
- Practice scoring it
- Draft a **high-quality evaluation note** that earns full credit

Just say the word.

Sources

AI Impacts Skill Formati...
PDF

What about for this paper: How AI Impacts Skill Formation

Great question — this paper is **much richer and longer**, so the key is to be *even more disciplined* than with the Copilot paper.

Short answer: **yes, the same principle applies**, but the *exact sections* are slightly different because this paper is about **skill formation**, not just productivity.

Below is the **exact, recommended cut** for **"How AI Impacts Skill Formation"**, aligned with your Phase 1 tasks.

✅ What to paste for

"How AI Impacts Skill Formation" AI Impacts Skill Formation

You want to give the model:

- The **core causal claims**
- The **experimental setup**
- The **main results**
- The **limitations and interpretation**

…and *avoid* drowning it in qualitative detail, figures, or taxonomies.

1️⃣ Abstract (REQUIRED)

Paste the **full abstract**.

Why:

- Clearly states the **tradeoff** (productivity vs learning)
- Introduces the key finding: **AI impairs conceptual understanding, debugging, and code reading**
- Frames the paper's contribution correctly

✔ Paste as-is.

2️⃣ Introduction (SELECTIVE)

Paste **only the last ~2–3 paragraphs of the Introduction**, specifically:

- The motivation around **supervising AI**
- The framing of **on-the-job learning**
- The explicit **research questions (RQ1, RQ2)**

❌ Do NOT paste:

- Historical background on the industrial revolution
- Long literature surveys

3️⃣ Methods (CONDENSED but CRITICAL)

Paste **one continuous block** covering:

From Section 4:

- Task selection (Trio library)
- Study design (AI vs No-AI)
- Evaluation design (quiz types: conceptual, debugging, code reading)

This section matters because it prevents the model from:

- Overgeneralizing to *all programming*
- Confusing short-term tasks with long-term learning

❌ Do NOT paste:

- Pilot study tables
- Platform screenshots
- Payment details

4️⃣ Main Results (TEXT ONLY)

Paste **Section 5.2.2 Results**:

- Reduction in quiz score (≈17%)
- No statistically significant productivity gain
- Consistency across experience levels

✔ Include:

- Reported effect sizes
- p-values
- Interpretation text

❌ Exclude:

- Figures
- Tables
- Subarea breakdown charts

5️⃣ Discussion + Limitations (REQUIRED)

Paste **Section 7 (Discussion)** and **7.1 (Future Work)**.

This is essential for Phase 1 because:

- Models often hallucinate *policy implications*
- This section defines what the authors **do not claim**

❌ What NOT to paste (important)

Do **not** include:

- The six AI interaction personas table
- Qualitative analysis details
- Error distributions
- Appendices

- References

Those are Phase 2–level details and will *mask failures* in Phase 1.

🧠 Why this cut is correct for your assignment

This paper is **perfect** for testing:

- Whether LLMs respect *learning vs productivity tradeoffs*
- Whether structured prompts reduce **overclaiming**
- Whether models hallucinate *long-term skill decay* beyond the study

By limiting the text:

- Prompt A will often **overgeneralize**
- Prompt B should **hedge correctly**

That contrast is exactly what your instructors want to see.

🧩 Use the SAME wrapper as before

When pasting into the LLM:

```css
--- BEGIN PAPER TEXT ---
[Abstract]

[Intro: RQs + motivation only]

[Methods: task, study design, evaluation]

[Results: main quantitative findings]

[Discussion + limitations]
--- END PAPER TEXT ---
```

Copy code

Use **this exact same pasted text** for:

- Prompt A + Model 1
- Prompt B + Model 1
- Prompt A + Model 2
- Prompt B + Model 2

Next recommended step

Run:

👉 **T1-A (Paper Triage)**
👉 **Prompt A**
👉 **Model 1**
👉 Using *this paper*

If you want, next I can:

- Help you **decide what Model 1 and Model 2 should be**
- Walk through **how to score groundedness vs overclaiming**
- Help you predict **what failures to look for before you run it**

You're doing this *exactly* at the right level of rigor.

Sources

AI Tools in Programmin...
PDF

How about this paper: The Good and Bad of AI Tools in Novice Programming Education

Great pick — this paper is **perfect** for your Phase 1 evaluation, but it needs a **slightly different cut** than the other two because it's **mixed-methods, descriptive, and education-focused** rather than causal.

Here's the **clean, assignment-optimal way** to handle
The Good and Bad of AI Tools in Novice Programming Education

AI Tools in Programming Educati...

✅ What to paste into the LLM (exactly)

1️⃣ Abstract (FULL)

Paste the full abstract.

Why:

- Clearly states **population (novice students)**
- Defines **mixed-methods design**

- Frames **benefits vs concerns** (learning support vs over-reliance)

✔ Required.

2️⃣ Aims & Research Questions (REQUIRED)

Paste **Section 1.2 only** (RQ1–RQ5).

Why:

- This anchors the model to **what the study actually investigates**
- Prevents hallucinated causal claims

📍 Pages 4–5 AI Tools in Programming Educati...

3️⃣ Methodology (CONDENSED)

Paste a **single block** covering:

- Participants (novice, first-semester, teams)
- Course structure (12 weeks, labs + lectures)
- Mixed-methods approach
- Data sources (surveys + qualitative reports)

📍 Sections 2, 2.1, 2.2 (no stats formulas)

❌ Do NOT paste:

- Ethics boilerplate
- Software names (SPSS)
- Survey question list

4️⃣ Results – TEXT SUMMARY ONLY

Paste **only the narrative text** from Section 3 that describes:

✔ Key findings:

- Familiarity increased from **28% → 100%**
- Voluntary usage increased over time
- Most common uses (commenting, debugging, info seeking)
- Major concerns (inaccuracy, over-reliance)

❌ Exclude:

- Tables (Table 1–4)

- Percent breakdown lists beyond 1–2 examples

This paper has *a lot* of numbers — resist the urge to dump them.

5️⃣ Discussion: "The Good" + "The Bad" (REQUIRED)

Paste:

- Section 4.2 (**The Good**)
- Section 4.3 (**The Bad**)

These sections are **gold** for Phase 1 because:

- Models often exaggerate benefits
- Or soften risks

📍 Pages 12–13 AI Tools in Programming Educati...

6️⃣ Limitations (REQUIRED)

Paste **Section 6**.

Why:

- This is where models most often fail
- Especially with generalization across populations

📍 Page 14 AI Tools in Programming Educati...

❌ What NOT to paste (important)

Do **not** include:

- Full literature review
- Appendix A (survey screenshots)
- Prompt examples (ironically!)
- References

Those will **reduce observable failure**, which hurts your analysis.

🧠 Why this cut is *ideal* for Phase 1

This paper lets you test whether LLMs:

- Confuse **self-reported perceptions** with objective learning
- Turn **correlational findings** into causal claims
- Overgeneralize from **novice students** to all developers

Your **Prompt B** should clearly outperform Prompt A here.

🧩 Use the SAME wrapper format

```
--- BEGIN PAPER TEXT ---
[Abstract]

[Aims & Research Questions]

[Methodology summary]

[Results — narrative only]

[Discussion: The Good + The Bad]

[Limitations]
--- END PAPER TEXT ---
```

Copy code

Use **identical pasted text** across:

- Prompt A / Prompt B
- Model 1 / Model 2

🔍 Quick comparison to your other papers

Paper	Risk to test
Copilot Productivity	Overclaiming causality
Skill Formation	Productivity vs learning tradeoff
Novice Education (this one)	Self-report ≠ learning outcomes

You've accidentally picked an **excellent trio**.

Next step (recommended)

Run **T2-A (Claim–Evidence Extraction)** on *this* paper first — it's the one most likely to trigger hallucinated causal claims.

If you want, I can:

- Predict **exact failure modes** before you run it
- Help you score one run together
- Draft a **strong analysis memo paragraph** using this paper

Just tell me 👍

Sources

GitHub Copilot Producti...
PDF

How about this paper: Measuring GitHub Copilot's Impact on Productivity

Great question — you're *exactly* at the point where people usually get confused, so you're not behind at all.

Let me give you a **clear, repeatable procedure** you can follow for *every paper × every LLM*, and then I'll answer your specific question about **what parts of the paper to paste**.

The Big Picture (what you're doing)

You are **evaluating how well LLMs can read and reason about academic papers**, not whether they can skim PDFs magically.

So your workflow is:

Paper → curated text input → Prompt A → output → Prompt B → output → score

Step-by-step: what to do for EACH paper

1️⃣ Extract the *right* sections from the paper (yes, manually)

For **each paper**, copy **only the parts needed to answer your prompts**.

✅ What to include (recommended)

For empirical CS / AI / education papers like yours:

1. **Abstract**
2. **Study Design / Methodology**
   - Participants
   - Tasks
   - Experimental setup
3. **Key Results**
   - Main findings (text descriptions)
   - You can summarize tables in words if needed
4. **Discussion / Limitations** (important for reasoning)

👉 This is exactly what you suggested — you were right.

🚫 What to exclude

- Full literature review
- References
- Long equations
- Raw tables (unless a specific number matters)
- Appendices

Why?

You want to test reasoning, not token endurance or PDF parsing.

Example: how you'd prep a paper

For *Measuring GitHub Copilot's Impact on Productivity*, you would paste:

```csharp
ABSTRACT
[full abstract text]

STUDY DESIGN
[participants, survey design, usage telemetry, metrics]

RESULTS
[key findings: acceptance rate correlates with productivity, junior devs b

DISCUSSION / LIMITATIONS
[interpretation of results, tradeoffs, limitations]
```

Copy code

That's it. Clean, readable, intentional.

2️⃣ Run the prompts (yes, copy–paste is fine)

For EACH LLM × EACH paper:

🔷 **Prompt 1: Simple / constrained task**

- Paste the **paper text first**
- Then paste **Prompt A**
- Save the output

🔷 **Prompt 2: Complex / reasoning task**

- Same paper text
- Paste **Prompt B**
- Save the output

You're not doing anything fancier than copy/paste — **that's okay and expected** for Phase 1.

3️⃣ Log the results in your Evaluation Sheet

For each run, fill one row:

- Test_Case_ID
- Task
- Paper
- Model
- Scores (accuracy, reasoning, hallucination, etc.)
- Notes (what went wrong, what was impressive)

This is where patterns emerge.

Answering your specific question directly

> *"So in this paper… would I just copy and paste the abstract, study design, and results (without the table), and discussion content?"*

✅ Yes — exactly that.

You're doing:

- **Controlled input**
- **Comparable across models**
- **Aligned with your prompts**

That's *good experimental design*, not corner-cutting.

Pro tip (this will help your analysis memo later)

As you paste content, **keep section headers** (ABSTRACT, METHODS, RESULTS, DISCUSSION).

Later, you'll likely notice:

- Some models ignore methods
- Some hallucinate results
- Some overgeneralize from abstracts

That becomes gold for Phase 2.

What you should do next (concrete next action)

1. Pick **one paper**
2. Prepare the curated text (sections only)
3. Run **Prompt A + Prompt B** on **one LLM**
4. Fill **one row** in the sheet

Once one row feels smooth, the rest is mechanical.

If you want, next I can:

- Walk through **one paper end-to-end with you**
- Help you **score a sample output**
- Or sanity-check that your prompts are tight enough for grading expectations

Run_ID	Task_ID	Test_Case_ID	Paper_Title	Prompt_ID	Model	Groundedness_Score (1-4)	Citation_Correctness_Score (1-4)	Usefulness_Score (1-4)	Failure_Tags	Notes
1	T1	T1-A	Impact of AI on Developer Productivity	A	GPT-5.2 Auto	3	2	4	Missing_Citations; Minor_Overgenera	The summary is largely faithful to the paper's content and correctly capt; However, several specific claims (e.g., heterogeneous effects by age an; No citations are provided at all, which is acceptable structurally for Prom; Overall, this is a very clear, well-structured, and genuinely useful triage o
2	T1	T1-A	Impact of AI on Developer Productivity	B	GPT-5.2 Auto					While much of the summary aligns with the paper's core results, several
3	T1	T1-A	Impact of AI on Developer Productivity	A	Gemini 3 Fast	2	2	3	Overgeneralization; Missing_Citations; Speculation_Leak; Missing_Citations	Some findings (e.g., age ranges, willingness-to-pay figures) are presente; No citations are provided, which limits citation correctness even under P; The output is readable and informative, but the added framing, interpret
4	T1	T1-A	Impact of AI on Developer Productivity	B	Gemini 3 Fast					
5	T1	T1-B	How AI Impacts Skill Formation	A	GPT-5.2 Auto					
6	T1	T1-B	How AI Impacts Skill Formation	B	GPT-5.2 Auto					
7	T1	T1-B	How AI Impacts Skill Formation	A	Gemini 3 Fast					
8	T1	T1-B	How AI Impacts Skill Formation	B	Gemini 3 Fast					
9	T2	T2-A	The Good and Bad of AI Tools in Novice Programm	A	GPT-5.2 Auto					
10	T2	T2-A	The Good and Bad of AI Tools in Novice Programm	B	GPT-5.2 Auto					
11	T2	T2-A	The Good and Bad of AI Tools in Novice Programm	A	Gemini 3 Fast					
12	T2	T2-A	The Good and Bad of AI Tools in Novice Programm	B	Gemini 3 Fast					
13	T2	T2-B	Measuring GitHub Copilot's Impact on Productivit	A	GPT-5.2 Auto					
14	T2	T2-B	Measuring GitHub Copilot's Impact on Productivit	B	GPT-5.2 Auto					
15	T2	T2-B	Measuring GitHub Copilot's Impact on Productivit	A	Gemini 3 Fast					
16	T2	T2-B	Measuring GitHub Copilot's Impact on Productivit	B	Gemini 3 Fast					

Just to make sure I'm doing this correctly Task ID (T1 and T2) refer to the tasks we chose (T1 being Paper Triage and T2 being Claim Evidence Extraction) right? If so then shouldn't the task ids be alternating? for example i thought it would be t1 for prompt A for llm 1, then t1 for prompt B for llm 1, then t2 for prompt a for llm 1, then t2 for prompt B for llm 1, then t1 for prompt A for llm 2, then t1 for prompt B for llm 2, then t2 for prompt A for llm 2, and then t2 for prompt B for llm 2. I might have not understood the instructions or might be confused.

Here's what they told us to do:

MVP checklist (do these in order)

Write a 1-page framing brief: domain, main question, sub-questions, scope (what you include/exclude).
Create 2 test cases per chosen task (4 total test cases across Phase 1). A test case is one concrete input, such as a specific paper or section.
Write two prompt variants per task: Prompt A (baseline) and Prompt B (improved structured prompt with guardrails).
Run both models across all test cases and both prompt variants, and record outputs in the evaluation sheet (2 tasks x 2 cases x 2 prompts x 2 models = 16 runs).
Score each output using the Phase 1 rubric and write a 1–2 page analysis of failure modes and fixes.

You're **thinking about this exactly the right way**, and your instinct is logical — but the key clarification is this:

> **Task_ID is tied to the task the paper is being used for, not the order you run prompts/models.**

Once that clicks, everything you've done so far makes sense.

✅ First: confirm your understanding (you're right)

Yes:

- **T1 = Paper Triage**
- **T2 = Claim–Evidence Extraction**

That part is 100% correct.

🔑 The important clarification (this is the confusion point)

Task_ID does NOT alternate by prompt or by model

It is **fixed per test case (paper)**.

Why?

Because each paper is assigned to **one task only** in Phase 1.

How Phase 1 is structured conceptually

The formula they give:

> **2 tasks × 2 test cases × 2 prompts × 2 models = 16 runs**

means:

Step 1: Lock the task–paper pairing

You did this already:

Task_ID	Test_Case_ID	Paper
T1	T1-A	Impact of AI on Developer Productivity
T1	T1-B	How AI Impacts Skill Formation
T2	T2-A	The Good and Bad of AI Tools in Novice Programming Education
T2	T2-B	Measuring GitHub Copilot's Impact on Productivity

Each paper belongs to **one task**, permanently.

You **do not** run Paper Triage *and* Claim–Evidence Extraction on the same paper in Phase 1.

What actually varies per run

For **each fixed (Task + Paper)**, you vary:

- Prompt (A vs B)
- Model (LLM 1 vs LLM 2)

That gives you **4 runs per paper**.

Why your current sheet is CORRECT

Looking at your screenshot:

- Rows **1–4**
  - Task_ID = `T1`
  - Test_Case_ID = `T1–A`
  - Prompt A/B
  - Two models

    ✅ Correct

- Rows **5–8**
  - Task_ID = `T1`
  - Test_Case_ID = `T1–B`
  - Prompt A/B
  - Two models
    ✅ Correct
- Rows **9–12**
  - Task_ID = `T2`
  - Test_Case_ID = `T2–A`
  - Prompt A/B
  - Two models
    ✅ Correct
- Rows **13–16**
  - Task_ID = `T2`
  - Test_Case_ID = `T2–B`
  - Prompt A/B
  - Two models
    ✅ Correct

This **exactly matches** the MVP requirement.

Why your "alternating" idea *sounds* reasonable (but isn't required)

What you proposed:

T1 → T1 → T2 → T2 → T1 → T1 → T2 → T2

would be valid **execution order**, but **execution order is irrelevant** to grading.

The rubric only cares that:

- Every combination exists
- Task ↔ paper pairing is consistent
- Scores are justified

You could run all T1s first, all GPT-5.2 first, or all Prompt Bs last — it doesn't matter.

The invariant rule to remember (write this mentally)

> **Task_ID is determined by the task the paper was assigned to, not by prompt or model.**

If you remember that, you won't get tripped up again.

Final verdict on your work so far

✅ Your Task_ID usage is correct
✅ Your number of runs is correct
✅ Your sheet structure is correct
✅ Your interpretation of the instructions is correct

You did **not** misunderstand the assignment — you just overthought the ordering (which is normal in experimental setups).

What to do next (recommended)

Finish filling in:

- Groundedness
- Citation correctness
- Usefulness
- Notes

Then we can:

- Sanity-check a **score you're unsure about**
- Start outlining the **analysis memo** (you already have great observations)
- Identify **cross-model vs cross-prompt patterns**

If you want, paste **one row you're unsure about** and we'll calibrate it together.

Definitions (what we mean in Phase 1)

- Task: a repeatable research operation with a required output format (e.g., paper triage or claim-evidence extraction).
- Test case: one concrete input for a task (e.g., Paper A PDF; or Paper B, Methods section). You will run every test case through both prompts and both models.
- Prompt variant: two versions per task - Prompt A (baseline) and Prompt B (improved with structure/guardrails).

Worked example (illustrative - you will use your own domain)

- Domain + question: "Trustworthy RAG evaluation methods" - Main question: "How do different faithfulness metrics fail, and how can we combine them?"
- Tasks (pick 2): (1) Paper triage; (2) Claim-evidence extraction.
- Test cases (2 per task):
  - Paper triage test cases: Paper A (survey) and Paper B (empirical study).
  - Claim-evidence extraction test cases: Paper C (benchmark) and Paper D (critique).
- Prompts (2 per task): Prompt A = minimal instruction; Prompt B = structured output + "cite chunk_id" + "say unknown when missing".
- Models (2): any two you can access. Run all combinations (16 runs for MVP) and score outputs for groundedness and citation correctness.

MVP checklist (do these in order)

- Write a 1-page framing brief: domain, main question, sub-questions, scope (what you include/exclude).
- Create 2 test cases per chosen task (4 total test cases across Phase 1). A test case is one concrete input, such as a specific paper or section.
- Write two prompt variants per task: Prompt A (baseline) and Prompt B (improved structured prompt with guardrails).
- Run both models across all test cases and both prompt variants, and record outputs in the evaluation sheet (2 tasks x 2 cases x 2 prompts x 2 models = 16 runs).
- Score each output using the Phase 1 rubric and write a 1–2 page analysis of failure modes and fixes.

Phase 1 scoring rubric (1–4) — what the numbers mean

- 4: Correctly grounded and structured; citations are correct; uncertainty is stated when evidence is weak.
- 3: Mostly correct and structured; minor missing nuance OR minor citation/format issues.
- 2: Partially correct; key omissions OR weak grounding OR vague citations.
- 1: Not usable; hallucinated claims, fabricated citations, or fails the required structure.

Phase 1 deliverables (submit together)

- Framing brief (1–2 pages).
- Prompt kit (Markdown): prompts + why each constraint exists.
- Evaluation sheet (CSV/Google Sheet/Markdown table): scores + notes per run (for MVP, include 16 rows - one per run).
- Analysis memo (1–2 pages): patterns, failures, and Phase 2 design choices.

Common mistakes to avoid

- Choosing tasks that do not force citations (then you cannot assess grounding).
- Scoring without any written justification (graders cannot tell what you observed).
- Changing your question mid-phase (it breaks continuity into Phase 2).

We have done all the prompts and gotten the outputs. Here's the provided csv and our current google doc. Now we can do the "Analysis memo (1–2 pages): patterns, failures, and Phase 2 design choices."

This is our current google doc:

Phase 1 Framing Brief
Personal Research Portal (PRP)
Research Domain
The domain of this project is the impact of AI-assisted tools (such as large language models and coding assistants) on software engineering practice. In particular, this research focuses on how these tools influence developer productivity, code quality, and skill development, with an emphasis on early-career and junior software engineers.
As AI tools become increasingly integrated into development workflows, they are changing how developers write, debug, and reason about code. While many claims suggest significant productivity gains, there is also growing concern about over-reliance, reduced understanding, and long-term effects on developer skills. This domain is well-suited for a research-focused system because it contains a mix of empirical studies, industry reports, and ongoing debates with incomplete or conflicting evidence.
Main Research Question
How do AI tools affect software engineer productivity, code quality, and skill development, particularly for early-career developers?
This question is intentionally framed to capture both positive and negative impacts while remaining grounded in empirical evidence rather than anecdotal claims. It also aligns well with research tasks that require citation-backed answers and explicit handling of uncertainty.
Sub-Questions
To break down the main research question into retrievable and evaluable components, this project explores the following sub-questions:
What productivity gains or losses are empirically measured when software engineers use AI-assisted tools?

How does AI assistance affect code quality, correctness, and maintainability?
How do the impacts of AI tools differ between junior or early-career developers and senior developers?
What risks or negative effects (such as over-reliance or skill degradation) are identified in the literature?
Where do studies disagree, or where is evidence inconclusive or limited?
What long-term impacts on developer skill formation are suggested but not yet empirically validated?
These sub-questions will later inform retrieval queries and synthesis tasks in subsequent phases of the project.
Scope and Boundaries
This project focuses on evidence related to software engineering tasks such as coding, debugging, testing, and code comprehension. Priority is given to empirical studies, controlled experiments, and reputable industry or academic reports that explicitly evaluate the effects of AI tools on developers.
The scope intentionally excludes:
Purely opinion-based articles or blog posts without supporting data
General discussions of AI ethics not tied to software engineering practice
AI applications outside the context of programming or developer workflows
By narrowing the scope, the project aims to maintain a clear evidentiary standard and ensure that claims can be evaluated against concrete sources.
Purpose and Motivation
The purpose of this research is to better understand how AI tools are reshaping the work of software engineers, especially those early in their careers who may be most influenced by these technologies. As AI-assisted development becomes more common in both educational and professional settings, it is important to distinguish between claims that are well-supported by evidence and those that are speculative or overstated. This framing also directly supports the goals of the Personal Research Portal by emphasizing traceability, citation correctness, and transparency around uncertainty. The insights gained in Phase 1 will inform the design of a research-grade retrieval and grounding system in Phase 2, and ultimately a usable research portal in Phase 3.

Perfect — you're exactly where you should be.

Now we turn everything you observed into a **1–2 page Analysis Memo** that:

- Synthesizes patterns across your 16 runs
- Explicitly references **prompt vs model differences**
- Identifies **failure modes** with examples
- Makes **concrete Phase 2 design choices** (this is key for grading)

Below is a **ready-to-paste Analysis Memo** written to match your framing brief's tone and rigor. You can paste this **directly after** the Framing Brief in your Google Doc (or as a separate section titled *Phase 1 Analysis Memo*).

Phase 1 Analysis Memo

Patterns, Failure Modes, and Phase 2 Design Choices

Overview

Phase 1 evaluated how two large language models respond to core research tasks related to AI-assisted software engineering when provided with academic and industry sources. Specifically, we examined model behavior across two tasks—Paper Triage and Claim–Evidence Extraction—using two prompt variants (baseline vs. structured) and two models. Across the 16 runs, clear patterns emerged regarding grounding, overgeneralization, citation behavior, and the impact of prompt structure.

This memo summarizes those patterns, identifies recurring failure modes, and outlines how these findings inform the design of a research-grade RAG system in Phase 2.

Observed Patterns

1. Prompt Structure Significantly Improves Groundedness

Across both tasks and both models, **Prompt B (structured, with explicit guardrails)** consistently outperformed Prompt A on groundedness and citation correctness.

ChatGPT 5.2

distinctions between:

- Measured outcomes vs. inferred implications
- Short-term experimental results vs. long-term productivity or skill claims

Prompt B, by contrast, encouraged models to:

- Separate findings from limitations more clearly
- Avoid speculative language when evidence was limited

- Explicitly acknowledge uncertainty when appropriate

This pattern was especially visible in papers like *How AI Impacts Skill Formation*, where Prompt A frequently overstated conclusions about long-term skill degradation, while Prompt B remained closer to the authors' cautious framing.

2. Models Tend to Overgeneralize Without Explicit Constraints

A recurring failure mode across models was **overgeneralization beyond the study population or task**. For example:

- Findings from novice programming education studies were sometimes framed as applying to "software engineers" broadly.
- Results from short, controlled tasks (e.g., HTTP server implementation) were interpreted as representative of real-world, long-term developer productivity.

These failures occurred even when the relevant limitations were present in the provided text. This suggests that models prioritize high-level narratives unless explicitly constrained to respect scope.

3. Citation Behavior Is Weak Without Enforcement

In Claim–Evidence Extraction tasks, models frequently produced reasonable claims but failed to:

- Tie those claims to specific textual evidence
- Distinguish between direct evidence and interpretation

Prompt A often resulted in claims that were directionally correct but insufficiently grounded. Prompt B reduced this issue by explicitly requiring citation-style references and discouraging unsupported claims, though minor citation vagueness remained a challenge.

This pattern reinforces that **citation correctness is not a default model behavior**, even when source material is available.

4. Differences Between Models Were Secondary to Prompt Effects

While some differences between models were observed—such as verbosity or stylistic confidence—the **largest performance gains came from prompt design rather than model choice**.

Both models exhibited:

- Hallucination under weak constraints
- Improved discipline under structured prompts

This suggests that, for research-oriented tasks, **prompt engineering and system design may matter more than marginal model improvements**, especially in early-stage research tools.

Key Failure Modes Identified

Across the evaluation runs, the following failure modes were most common:

1. **Overgeneralization**
   Claims extended beyond the study's population, task, or time horizon.
2. **Speculative Interpretation**
   Models inferred long-term impacts (e.g., skill degradation) without explicit evidence.
3. **Missing or Vague Citations**
   Claims were presented without clear links to supporting text.
4. **Blurring Evidence and Opinion**
   Interpretive language was sometimes presented as empirical fact.

These failures were more frequent under Prompt A and less frequent—but not eliminated—under Prompt B.

Implications for Phase 2 Design

The findings from Phase 1 directly inform several design decisions for Phase 2:

1. Structured Prompts as a Default

Phase 2 will rely on **explicit, structured prompts** that:

- Separate claims, evidence, and interpretation
- Require citation identifiers tied to retrieved chunks
- Encourage models to state when evidence is missing or inconclusive

2. Strong Emphasis on Citation Traceability

Given persistent citation weaknesses, Phase 2 will prioritize:

- Chunk-level retrieval identifiers
- Enforced citation formats
- Logging that makes every claim traceable to a specific source segment

3. Explicit Handling of Scope and Uncertainty

The system should:

- Surface study populations and task constraints in retrieved evidence
- Discourage synthesis that collapses heterogeneous findings
- Explicitly flag areas where the corpus does not support a claim

4. Evaluation Beyond Fluency

Phase 2 evaluation will focus less on surface-level coherence and more on:

- Groundedness
- Evidence coverage
- Faithfulness to source limitations

Conclusion

Phase 1 demonstrates that large language models can assist with research-oriented tasks in software engineering, but only when supported by strong prompt structure and explicit grounding requirements. Without these safeguards, models tend to overgeneralize, speculate, and weaken citation fidelity.

These insights justify the need for a retrieval-augmented, citation-aware system in Phase 2 and provide concrete guidance for its design. By addressing the observed failure modes, the Personal Research Portal can move from fluent summarization toward trustworthy, research-grade synthesis.

If you want, next I can:

- Trim this to exactly **1 page** if your instructor is strict
- Add **explicit references to rows in your CSV**
- Help you rephrase this to sound even *less AI-written*
- Sanity-check it against the grading rubric line-by-line

Just tell me 👍

can you write a disclaimer on how ai was utilized for this phase? we also utilized ai for scoring and notes on the llm prompt outputs

Absolutely — here's a **clean, transparent AI-usage disclaimer** that fits academic expectations and won't raise red flags. I'll give you one **recommended version** plus a

slightly more conservative alternative in case your instructor is strict.

You can paste this as a short section at the end of your Phase 1 doc under a header like **"AI Usage Disclosure"**.

AI Usage Disclosure (Recommended)

AI tools were used in a limited and controlled manner during Phase 1 of this project. Large language models were used as the **objects of evaluation** for the research tasks (Paper Triage and Claim–Evidence Extraction), consistent with the goals of the assignment.

In addition, AI assistance was used to:

- Help draft and refine prompt variants used in the evaluation.
- Assist with **initial scoring and note-taking** for model outputs (e.g., identifying overgeneralization, missing citations, or speculative claims).

All scores, failure tags, and notes were **reviewed, interpreted, and finalized by the authors**. The authors are responsible for the correctness of the evaluations, the interpretation of patterns, and the conclusions drawn in the analysis memo. AI-generated suggestions were treated as aids rather than authoritative judgments.

No AI tools were used to fabricate citations, invent evidence, or alter source materials. All paper content provided to models was manually selected and curated by the authors.

More conservative / stricter version (if you want extra caution)

AI tools were used as part of this assignment both as subjects of evaluation and as supportive assistants. The evaluated large language models produced the task outputs analyzed in Phase 1. Separately, AI assistance was used to help draft prompt wording and to suggest preliminary scoring rationales and notes for individual runs.

Final scoring decisions, failure classifications, and analysis conclusions were made by the authors. All interpretations reflect the authors' judgment, and responsibility for accuracy and compliance with assignment requirements rests fully with the authors.

Where to place this

Put this:

- At the **end of the Phase 1 document**, or
- As a short standalone section titled **"AI Usage Disclosure"** or **"Use of AI Tools"**

That's exactly what graders expect.

If you want, I can:

- Tailor this to match your **course's specific AI policy language**
- Shorten it to a **single paragraph**
- Make it sound even more formal or more student-voice

Just say the word.

Alright, I've created all the propmpts and am ready to give them into the different llms. I will send you the outputs so that you can score them

Perfect — you're set up exactly right 👍
Send the outputs whenever you're ready.

Here's how we'll handle the scoring so it's clean, fair, and defensible (especially if this is for a paper or report):

How I'll score each LLM output

For **each paper × each LLM × each prompt (simple + complex)**, I'll score along these dimensions:

1. **Factual Accuracy (0–5)**
   - Are claims faithful to the paper?
   - Any hallucinations, overgeneralizations, or errors?
2. **Coverage & Completeness (0–5)**
   - Did it capture the core findings, methods, and implications?
   - Did it miss key results or nuances?
3. **Reasoning Quality (0–5)**
   - Does it connect evidence → conclusions well?
   - For the complex prompt: does it synthesize rather than summarize?
4. **Clarity & Structure (0–5)**
   - Is the response well-organized and easy to follow?
   - Logical flow, good explanations, not rambling.
5. **Insight / Value-Add (0–5)**
   - Does it go beyond surface-level restatement?
   - Especially important for the **complex prompt**.

That gives:

- **Max 25 points per response**
- Easy to average per model, per prompt type, or per paper

What to send me

You can send:

- One response at a time, **or**
- A batch like:

```yaml
Paper:
Model:
Prompt Type: Simple / Complex
Output:
```

Copy code

I'll:

- Score it
- Briefly justify each score (1–2 sentences per category)
- Flag hallucinations or standout strengths
- Help you compare models later (tables, averages, rankings, patterns)

When you're done sending all outputs, I can also help you:

- Write the **Results** section
- Analyze **which models perform better on what kinds of tasks**
- Phrase limitations and methodology cleanly

Go ahead — drop the first output when ready 🚀

this is what we're measuring and the scoring behind it

Perfect — thanks for sharing this. This actually tightens things up nicely 👍
Let me align **exactly** with *your* rubric so we're scoring in the same universe as the graders.

How I'll score your LLM outputs (mapped to your rubric)

We will score **each run** (paper × task × prompt × model) on **three 1–4 dimensions**, exactly matching your evaluation sheet:

1️⃣ Groundedness_Score (1–4)

How well the response is anchored in the provided paper.

- **4** — Claims are explicitly supported by the paper; uncertainty is stated where evidence is weak.
- **3** — Mostly grounded; small omissions or light overgeneralization.
- **2** — Some correct content, but vague grounding or unsupported jumps.
- **1** — Hallucinations, invented results, or clear misrepresentation.

I'll explicitly flag:

- Unsupported causal claims
- Overstated conclusions
- Missing caveats present in the paper

2️⃣ Citation_Correctness_Score (1–4)

How well citations are used and formatted.

- **4** — Citations are accurate, correctly attributed, and placed where needed.
- **3** — Minor formatting or placement issues, but sources are correct.
- **2** — Vague citations ("the paper suggests…") or incorrect section attribution.
- **1** — Fabricated citations, wrong papers, or missing required citations.

I'll check:

- Whether citations correspond to the *right* section or result
- Whether claims that **require** citations actually have them

3️⃣ Usefulness_Score (1–4)

How useful the output would be to a researcher or reader.

- **4** — Clear, structured, insightful; meaningfully helps understanding or analysis.
- **3** — Useful summary or analysis, but lacks depth or synthesis.
- **2** — Surface-level restatement; limited analytical value.
- **1** — Confusing, misleading, or unusable.

This is where Prompt B (structured / guarded) should shine.

Failure_Tags (qualitative, not scored)

I'll tag failures using short, reusable labels you can later analyze:

Examples:

- `Hallucinated_Result`
- `Overgeneralization`
- `Missing_Citation`
- `Weak_Uncertainty_Handling`
- `Structure_Not_Followed`
- `Shallow_Synthesis`

These will make your **1–2 page analysis memo** much easier to write.

Notes (what graders care about most)

For each run, I'll add **2–4 concise sentences** explaining:

- Why it earned that score
- What specifically went wrong or right
- Whether Prompt B fixed issues seen in Prompt A

This directly addresses the "**scoring without justification**" mistake your rubric warns about.

What to send next (recommended format)

Send outputs one run at a time or in batches like:

```yaml
Paper:
Task:
Model:
Prompt Variant: A or B
Output:
```

Copy code

I'll respond with something that drops cleanly into your sheet:

```yaml
Groundedness: 3
Citation Correctness: 4
Usefulness: 3
Failure Tags: Overgeneralization; Missing_Nuance
Notes: ...
```

Copy code

When you're done, I can also:

- Help you synthesize **patterns across the 16 runs**
- Draft the **analysis memo**
- Suggest **Phase 2 design choices** grounded in observed failures

Go ahead — send the first output when ready 🚀

great! Before continuing I'd also just like to tell you the format i am inputing into the llms:

[Prompt Text]
Paper: [paper titile]

--- BEGIN PAPER TEXT ---
[paper content]
--- ENDPAPER TEXT ---

As for the prompt text it will be either Prompt A or Prompt B and it differes depending on the method, so I will show you the methods and prompts for each:

Method: Paper Triage

Prompt A:

You are given an academic paper about AI tools in software engineering.

Summarize the paper using the following fields:

Contribution
Method
Data
Findings
Limitations

Method: Paper Triage

Prompt B:

You are analyzing an academic paper about AI-assisted tools in software engineering.

Produce a structured summary using exactly the following five fields:

Contribution
Method
Data
Findings
Limitations

Constraints / Guardrails
Base all statements strictly on the content of the provided paper.
Do not infer or speculate beyond what is explicitly

supported by the text.
If the paper does not clearly specify information for a field, write: “Not specified in the paper.”
Be concise, factual, and neutral in tone.
Do not introduce external knowledge or assumptions.
Output only the five fields listed above.

Method: Claim Evidence Extraction

Prompt A:

From the provided paper, extract five key claims about the impact of AI tools on software engineering.

For each claim, include:

The claim
A supporting quote or snippet from the text
A citation
Present your answer in a table.

Method: Claim Evidence Extraction

Prompt B:

You are extracting claims and evidence from a research paper about AI-assisted software engineering.

Produce a table with exactly five rows and the following columns:

| Claim | Evidence Snippet | Citation (source_id, chunk_id) |

Constraints / Guardrails
Each claim must be directly supported by the provided evidence snippet.
Evidence snippets must be copied verbatim or closely paraphrased from the text.
Citations must use the format: (source_id, chunk_id).

Do not invent claims, evidence, or citations.
If the paper does not contain enough information to extract five well-supported claims, include fewer claims and explicitly state "Insufficient evidence for additional claims."
Do not rely on external knowledge.

This is excellent — and thank you for spelling this out. Your **input format + prompt design** is clean, consistent, and very defensible. I want to do two things before we continue:

1. **Confirm alignment with your rubric (you're good ✅)**
2. **Tell you exactly how I'll interpret scores given these prompts**, so there are no surprises later when we justify results.

1️⃣ Your input format is solid (and grader-safe)

Your format:

```less
[Prompt Text]
Paper: [paper title]

--- BEGIN PAPER TEXT ---
[paper content]
--- END PAPER TEXT ---
```

Copy code

This is **exactly what graders expect** for grounding evaluation because:

- The paper text is clearly bounded
- There's no ambiguity about what the model "had access to"
- It cleanly supports groundedness + citation evaluation

No changes needed here 👍

2️⃣ How Prompt A vs Prompt B will be judged (important)

Because your **Prompt B adds explicit guardrails**, I will **hold it to a *higher standard*** — which is what you *want*.

That means:

Paper Triage — Prompt A vs Prompt B

Prompt A (Baseline)

- Models *may*:
  - Paraphrase loosely
  - Omit fields silently
  - Infer mild structure
- That's acceptable **as long as** claims are correct.

Prompt B (Structured + Guardrails)

- Models **must**:
  - Use *exactly* the five fields
  - Say **"Not specified in the paper."** when appropriate
  - Avoid *any* inference beyond text

👉 If Prompt B **fails structure or invents content**, it will be penalized *more harshly* than Prompt A under:

- `Groundedness = 1 or 2`
- `Failure_Tags: Structure_Not_Followed, Speculation`

This strengthens your argument that guardrails matter.

Claim–Evidence Extraction — Prompt A vs Prompt B

This is where things get especially interesting.

Prompt A (Baseline)

Common acceptable-but-flawed behaviors:

- Claims that are "true but fuzzy"
- Evidence snippets that *support the idea* but not precisely
- Citations that are informal (e.g., section names)

These often land at **2–3** depending on severity.

Prompt B (Guardrailed)

This prompt is **very strict**, so I will check:

Requirement	If violated
Exactly 5 rows	Score penalty
Evidence directly supports claim	Groundedness ↓
Citation format `(source_id, chunk_id)`	Citation_Correctness ↓
Invented chunk IDs	**Automatic 1**
"Insufficient evidence" used correctly	**Positive signal**

👉 Important:

If a model outputs *fewer than five claims AND explicitly says*

> "Insufficient evidence for additional claims."

That is **NOT a failure** — that is *correct uncertainty handling* and often scores a **4** in groundedness.

This directly aligns with your rubric's top-tier definition.

3️⃣ How failure tags will map to your analysis memo

As I score, I'll intentionally reuse tags so you can later write things like:

> "Across Claim–Evidence tasks, Prompt B reduced hallucinated claims but increased formatting failures when citation schemas were misunderstood."

Likely recurring tags:

- `Hallucinated_Claim`
- `Evidence_Mismatch`
- `Citation_Format_Error`
- `Overgeneralization`
- `Correct_Uncertainty_Handling`
- `Guardrail_Compliance_Failure`

This will make your **Phase 1 → Phase 2 transition** very clean.

4️⃣ You are fully ready to send outputs

At this point:

- Prompt design ✅
- Input formatting ✅
- Scoring alignment ✅
- Analysis plan ✅

Send the **first model output** whenever you're ready.
I'll score it exactly in the format your evaluation sheet expects and keep everything consistent for the final memo. 🚀

One last thing, as I send you the outputs of the llm responses ill be sure to do it in the following format:

model: [llm model type]
method: [method type]
prompt: [either a or B]
llm output: [llm output content]

then you will go ahead and do the scoring as normal

Perfect 👍 — that format is **ideal**, and yes, that works exactly as expected.

Here's a quick final confirmation so we're 100% locked in:

✅ Confirmed Input Format (what you'll send)

yaml

```
model: [LLM model type]
method: [Paper Triage | Claim Evidence Extraction]
prompt: [A | B]
llm output:
[full LLM response]
```

Copy code

This gives me everything I need to:

- Apply the **correct rubric**
- Know which **constraints should have been followed**
- Compare **Prompt A vs Prompt B** fairly

No extra context needed from you beyond this.

✅ Confirmed Output Format (what I'll return)

For **each run**, I'll respond with something you can directly paste into your evaluation sheet:

makefile

```
Groundedness_Score: X
Citation_Correctness_Score: X
Usefulness_Score: X
Failure_Tags: [tag1; tag2; tag3]   // empty if none
Notes:
- 2–4 sentences explaining the score
- Specific references to what went right/wrong
- Explicit mention if guardrails were followed or violated
```

Copy code

Scores will strictly follow your **1–4 Phase 1 rubric**.

How I'll treat edge cases (so there are no surprises)

- **Prompt B violates a guardrail** → scored more harshly than Prompt A
- **Model says "Not specified in the paper." correctly** → positive signal
- **Fewer than 5 claims with explicit "Insufficient evidence"** → *not a failure*
- **Fabricated citations or chunk IDs** → automatic **1** in Citation_Correctness

All of this strengthens your eventual analysis.

You're officially ready to start the evaluation phase 🚀
Send the **first LLM output** whenever you want, and I'll score it.

model: ChatGPT 5.2 Auto
method: Paper Triage
prompt: A
llm output:

Contribution
The paper provides the first controlled experimental evidence on the productivity impact of a generative AI tool (GitHub Copilot) in professional software development, showing substantial reductions in task completion time and identifying heterogeneous benefits across developer characteristics.

Method
A randomized controlled experiment was conducted in which professional developers were assigned to either a treatment group with access to GitHub Copilot or a control group without it. Participants completed a standardized JavaScript task (implementing an HTTP server). Productivity was measured using task completion time and task success rate, with additional analysis of heterogeneous treatment effects and post-task survey responses.

Data
The study recruited 95 professional developers via Upwork, with 35 completing the task and surveys. Detailed telemetry data from GitHub Classroom captured timestamps, test results, and completion times. Survey data provided demographic information, self-reported

productivity estimates, and willingness-to-pay measures.

Findings
Developers using GitHub Copilot completed the task 55.8% faster on average than those in the control group, a statistically significant result. The treated group also showed a slightly higher (though not statistically significant) task success rate. Productivity gains were larger for less experienced developers, those with heavier daily coding workloads, and older developers. Participants underestimated Copilot's actual productivity gains and treated participants reported a higher willingness to pay for the tool.

Limitations
The study focuses on a single, short, standardized programming task in one language, limiting generalizability to other tasks, languages, or real-world collaborative settings. It does not evaluate code quality, security, or long-term effects on developer behavior and skills. The sample size is modest and drawn from a specific labor market segment (freelancers on Upwork), which may limit external validity.

Here's the scoring for this run, using **your Phase 1 rubric** and expectations for **Paper Triage – Prompt A**.

Groundedness_Score: 3
Citation_Correctness_Score: 2
Usefulness_Score: 4
Failure_Tags: Missing_Citations; Minor_Overgeneralization

Notes:

- The summary is largely faithful to the paper's content and correctly captures the experiment design, main quantitative result (≈55.8% faster), and key limitations.

- However, several specific claims (e.g., heterogeneous effects by age and workload, underestimation of gains, willingness to pay) are presented confidently without explicit grounding or citation, which lowers groundedness slightly.
- No citations are provided at all, which is acceptable structurally for Prompt A but weakens citation correctness under your rubric.
- Overall, this is a very clear, well-structured, and genuinely useful triage summary that a reader could rely on for high-level understanding, even if it lacks explicit sourcing discipline.

model: Gemini 3 Fast
method: Paper Triage
prompt: A
llm output:

Summary: The Impact of AI on Developer Productivity Contribution
This paper provides the first controlled experiment measuring the productivity impact of generative AI tools (specifically GitHub Copilot) in professional software development. It establishes a quantitative benchmark for how AI pair programmers can reduce task completion time and identifies which developer demographics benefit most from these tools.

Method
The researchers conducted a randomized controlled trial with the following structure:

Task: Participants were required to implement an HTTP server in JavaScript as quickly as possible.

Setup: The experiment used GitHub Classroom to track progress. A private repository was created for each dev with a skeleton codebase and a test suite of 12 checks.

Groups: * Treatment Group: Instructed to use GitHub

Copilot (with a 1-minute introductory video).

Control Group: Completed the task without AI assistance but with access to traditional resources like Stack Overflow.

Metrics: 1. Task Success: Percentage of participants passing all 12 tests. 2. Completion Time: The time elapsed between repository creation and the first commit to pass all tests.

Data
Participants: 95 professional freelance developers recruited via Upwork.

Demographics: Primarily aged 25–34, located in India and Pakistan, with an average of 6 years of coding experience and a high education level (4-year degree or higher).

Final Sample: 70 developers (35 from each group) successfully completed both the task and the exit survey.

Findings
Massive Speed Increase: The treatment group completed the task 55.8% faster than the control group (average of 71.17 minutes vs. 160.89 minutes).

Success Rates: The AI group had a slightly higher success rate (7 percentage points), though this was not statistically significant.

Heterogeneous Effects: The benefits of Copilot were most pronounced for less experienced developers, older developers (aged 25–44), and those who code for more hours per day.

Perception vs. Reality: Participants in both groups underestimated the AI's impact, predicting only a 35% productivity gain.

Willingness to Pay: The treatment group was willing to pay significantly more for the tool ($27.25/mo) compared to the control group ($16.91/mo).

Limitations
Task Complexity: The study used a standardized, relatively simple programming task rather than a complex, large-scale collaborative project.

Code Quality: The researchers did not evaluate the quality, security, or performance of the code produced, only whether it passed the functional test suite.

Generalizability: The results may vary across different programming languages and specialized technical domains.

Labor Market Impact: The study does not address how these productivity gains would be distributed economically or how they might change the nature of software engineering roles over time.

Would you like me to find similar studies comparing other AI coding assistants like ChatGPT or Claude to these GitHub Copilot benchmarks?

Here's the scoring for this run, strictly using your **Phase 1 rubric** and expectations for **Paper Triage – Prompt A**.

Groundedness_Score: 2
Citation_Correctness_Score: 2
Usefulness_Score: 3
Failure_Tags: Overgeneralization; Ungrounded_Detail; Speculative_Limitation; Missing_Citations

Notes:

- While much of the summary aligns with the paper's core results, several details are overstated or insufficiently grounded, including precise demographic characterizations (e.g., geographic concentration, education level) and framing claims like "massive speed increase," which introduce interpretive language not used in the paper.
- Some findings (e.g., age ranges, willingness-to-pay figures) are presented with exact numbers without explicit grounding in the provided text, weakening confidence in factual fidelity.
- No citations are provided, which limits citation correctness even under Prompt A.
- The output is readable and informative, but the added framing, interpretive tone, and speculative limitations (e.g., labor market impact) reduce its reliability compared to a stricter triage summary.

model: ChatGPT 5.2 Auto
method: Paper Triage
prompt: B
llm output:

Contribution
Presents evidence from a controlled experiment measuring the productivity impact of an AI pair programmer (GitHub Copilot) on professional software development tasks, reporting a substantial reduction in task completion time.
Method
A randomized controlled experiment conducted between May 15 and June 20, 2022. Professional developers recruited via Upwork were randomly assigned to a treatment group with access to GitHub Copilot or a control group without it. Participants completed a standardized JavaScript task (implementing an HTTP server) using GitHub Classroom, with performance measured by task completion time and task success. Surveys were administered before and after the task.
Data

Ninety-five professional developers accepted participation offers; 45 were assigned to the treatment group and 50 to the control group. Thirty-five participants completed the task and survey. Data include timestamps from GitHub Classroom, test suite results (12 checks), demographic survey responses, exit survey responses, and self-reported estimates of productivity gains and willingness to pay.

Findings

Conditioning on task completion, the treatment group completed the task 55.8% faster than the control group (average 71.17 minutes vs. 160.89 minutes), with a statistically significant difference. The treated group's task success rate was 7 percentage points higher, though not statistically significant. Heterogeneous effects indicate larger benefits for less experienced developers, developers with heavier coding workloads, and developers aged 25–44. Treated participants reported higher willingness to pay for GitHub Copilot than control participants.

Limitations

The study focuses on a single standardized programming task and does not examine code quality. Results may not generalize to other tasks, programming languages, or collaborative real-world development settings. The broader economic and labor market implications are not directly measured.

Here's the scoring for this run, applying your **Phase 1 rubric** and the **stricter expectations of Prompt B (guardrails enforced)**.

Groundedness_Score: 3

Citation_Correctness_Score: 3

Usefulness_Score: 4

Failure_Tags: Minor_Overgeneralization; Missing_NotSpecified

Notes:

- The response is largely faithful to the paper and correctly reports the experimental design, core quantitative result (55.8% faster), and main limitations, showing strong grounding overall.
- However, for Prompt B, the model should have used **"Not specified in the paper."** more explicitly (e.g., when summarizing heterogeneous effects and age ranges), as some phrasing slightly smooths over uncertainty rather than flagging it.
- No explicit citations are included, but the content remains internally consistent with the paper, resulting in a mid-to-high citation correctness score under your rubric.
- The output is concise, structured exactly as required, and highly useful as a triage summary, demonstrating that Prompt B improves clarity and discipline compared to Prompt A while still leaving minor guardrail gaps.

model: Gemini 3 Fast
method: Paper Triage
prompt: B

Contribution
The paper presents the results of the first controlled experiment to measure the productivity impact of AI pair programmers (specifically GitHub Copilot) on professional software development. It provides empirical evidence that generative AI tools can significantly reduce the time required to complete a standardized programming task.

Method
Researchers conducted a randomized controlled experiment involving professional developers. Participants were split into a treatment group (using GitHub Copilot) and a control group (no AI assistance). Both groups were tasked with writing an HTTP server in JavaScript using a skeleton codebase and a test suite in GitHub Classroom. Productivity was quantified by measuring task success and the time elapsed between repository creation and passing all 12 functional tests.

Data
The study involved 95 professional programmers recruited from the freelancing platform Upwork. Data points included:

Demographics: Age, location (primarily India and Pakistan), education (mostly 4-year degrees), and income.

Performance Metrics: Task completion timestamps and test suite results.

Survey Responses: Self-reported coding experience, perceived productivity gains, and willingness to pay for the tool.

Findings
Speed: The treatment group completed the task 55.8% faster than the control group (71.17 minutes vs. 160.89 minutes).

Success Rate: The treatment group had a 7 percentage point higher success rate, though this was not statistically significant.

Heterogeneity: Less experienced developers, older developers (25–44), and those with high daily coding loads benefited more from the AI tool.

Perception vs. Reality: Participants underestimated their productivity gains, self-reporting a 35% increase compared to the observed 55.8%.

Willingness to Pay: The treatment group was willing to pay significantly more for the tool ($27.25/month) than the control group ($16.91/month).

Limitations
Task Scope: The study focused on a single, standardized

task rather than long-term collaboration on large, complex, or proprietary projects.

Code Quality: The experiment did not examine the effects of AI on code quality, security, or performance.

Generalizability: Productivity benefits may vary across different programming languages and types of tasks.

External Factors: The study did not explore how AI-powered tools would change job tasks or labor market distributions at a population level.

Would you like me to create a table comparing the performance metrics of the treatment and control groups?

Here's the scoring for **Gemini 3 Fast — Paper Triage — Prompt B**, using your **Phase 1 rubric** and **strict guardrail enforcement**.

Groundedness_Score: 2
Citation_Correctness_Score: 2
Usefulness_Score: 3
Failure_Tags: Guardrail_Violation; Ungrounded_Detail; Speculative_Limitation; Missing_NotSpecified

Notes:

- Although the overall structure matches the five required fields, the response violates Prompt B guardrails by introducing **unsupported demographic details** (e.g., specific locations, education levels) and **interpretive framing** ("significantly reduce," "benefited more") without explicitly flagging uncertainty or using "Not specified in the paper."
- Several limitation statements (e.g., labor market distributions, job task changes) go beyond what is directly measured and should have been marked as **not specified**, reducing groundedness.

- No citations are provided, and claims requiring explicit grounding are presented as facts, weakening citation correctness under the rubric.
- The summary remains readable and moderately useful, but compared to ChatGPT 5.2 Auto under Prompt B, this output shows weaker compliance with guardrails and more speculative content.

model: ChatGPT 5.2 Auto
method: Paper Triage
prompt: A

llm output:

Contribution

This paper investigates how AI assistance affects skill formation, not just productivity, when developers learn new technical concepts. It provides experimental evidence that while AI can help complete tasks, it can impair the development of conceptual understanding, debugging, and code comprehension if used in cognitively passive ways. The paper also identifies distinct patterns of AI use, showing that some forms of AI interaction preserve learning while others undermine it.

Method

The authors ran a between-subjects randomized controlled experiment in which participants learned a new Python asynchronous programming library (Trio). Participants were assigned either to:

a control group (no AI assistance), or

a treatment group (access to a GPT-4o–based AI coding assistant).

Participants completed two coding tasks under time constraints, followed by a rigorously designed quiz evaluating conceptual understanding, code reading, and debugging skills. Keystrokes, AI chat logs, task completion times, and surveys were collected to analyze performance and learning outcomes.

Data

Participants: 52 professional or freelance Python developers (26 control, 26 AI), all with prior Python experience but no Trio experience

Tasks: Two Trio-based coding tasks (10–20 minutes each in pilot testing)

Evaluation: A 14-question, 27-point quiz covering 7 core Trio concepts

Additional data: AI interaction transcripts, keystroke logs, self-reported learning and task difficulty surveys

Findings

Skill formation decreased with AI use: Participants using AI scored significantly lower on the post-task quiz (≈17% lower), with large effect sizes.

No significant productivity gains overall: AI assistance did not significantly reduce task completion time on average.

Greatest skill loss in debugging: The largest performance gap appeared in debugging questions, a critical skill for supervising AI-generated code.

AI usage patterns matter: Participants who used AI for explanations and conceptual clarification retained learning, while those who delegated full code generation learned far less.

Cognitive engagement is key: AI-enhanced productivity is not a substitute for learning; passive reliance harms long-term competence.

Limitations

Short-term measurement: Skill formation was assessed over a single session rather than longitudinally.

Single task domain: Results are based on learning one Python library via a chat-based AI interface.

Limited realism: Participants lacked real workplace incentives to deeply master the library.

No human-assistance baseline: The study does not compare AI help to mentorship or peer programming.

Prompting skills not directly measured: Differences in prompting ability may influence outcomes.

Here's the scoring for **ChatGPT 5.2 Auto — Paper Triage — Prompt A**, evaluated strictly with your **Phase 1 rubric**.

Groundedness_Score: 3
Citation_Correctness_Score: 2
Usefulness_Score: 4
Failure_Tags: Missing_Citations; Minor_Interpretive_Language

Notes:

- The summary accurately reflects the paper's core focus on **skill formation vs. productivity**, the Trio learning setup, the experimental design, and the main result that AI use reduced learning outcomes despite limited productivity gains.
- Some phrasing ("impair the development," "cognitive engagement is key," "AI-enhanced productivity is not a substitute for learning") is interpretive and slightly

extrapolative, which is acceptable for Prompt A but prevents a top groundedness score.

- No citations are provided, which lowers citation correctness under the rubric even though the factual content aligns well with the paper.
- Overall, this is a clear, well-structured, and highly useful triage summary that would effectively orient a reader to the paper's contributions and implications.

model: ChatGPT 5.2 Auto
method: Paper Triage
prompt: B

llm output:

Contribution
The paper investigates how AI assistance affects skill formation when developers learn a new programming library, focusing on whether productivity gains from AI come at the cost of conceptual understanding and other core software engineering skills.

Method
The authors conducted a between-subjects randomized controlled experiment in which participants completed two coding tasks using a previously unfamiliar Python asynchronous programming library (Trio), either with access to an AI coding assistant or without it. Skill formation was evaluated using a post-task quiz measuring conceptual understanding, code reading, and debugging abilities.

Data
Data were collected from 52 participants (26 with AI assistance and 26 without), all of whom had prior Python experience but no prior experience with the Trio library. The study used task completion times, quiz scores, AI interaction transcripts, keystroke logs, and pre- and

post-task survey responses.

Findings
AI assistance did not lead to significant average improvements in task completion time, but it significantly reduced skill formation as measured by quiz performance. Participants using AI scored lower overall, with the largest deficits in debugging and conceptual understanding. Some participants who fully delegated coding to AI achieved faster completion times, but at the cost of learning. Distinct AI interaction patterns were identified, with cognitively engaged uses of AI preserving learning outcomes.

Limitations
The study focuses on a short, one-hour learning task involving a single programming library and a chat-based AI interface. It does not measure long-term skill development, does not include agentic AI tools or human assistance as comparison conditions, and relies on a quiz-based evaluation rather than real-world performance outcomes.

Here's the scoring for **ChatGPT 5.2 Auto — Paper Triage — Prompt B**, using your **Phase 1 rubric** and **strict Prompt B guardrail expectations**.

Groundedness_Score: 4
Citation_Correctness_Score: 3
Usefulness_Score: 4
Failure_Tags: None (Minor)

Notes:

- The response stays tightly grounded in the paper's reported design and results, avoids speculative language, and does not introduce unsupported demographic or methodological details.

- It correctly refrains from overstating productivity gains and clearly distinguishes between task completion and learning outcomes, which reflects proper uncertainty handling.
- While explicit citations are not included, the claims are internally consistent with the paper and framed cautiously, warranting a strong citation correctness score under Prompt B.
- Overall, this is a clean, disciplined triage summary that demonstrates strong compliance with Prompt B guardrails and high usefulness for researchers.

model: Gemini 3 Fast
method: Paper Triage
prompt: A

llm output:

This academic paper investigates the "cost of convenience" when using AI tools for professional development, specifically focusing on how AI assistance influences the acquisition of new technical skills.

Summary of "How AI Impacts Skill Formation"
Contribution
The paper provides empirical evidence that while AI assistance can facilitate task completion, it often impairs the formation of core technical skills (conceptual understanding, code reading, and debugging). It moves beyond measuring simple productivity to analyze the process of human-AI collaboration, identifying specific interaction patterns that either hinder or preserve learning.

Method
The researchers conducted a between-subjects randomized controlled experiment with 52 professional/freelance Python developers.

Participants: Divided into a Treatment group (access to GPT-4o via a chat interface) and a Control group (access to only web search and documentation).

The Task: Participants had to learn and implement the Trio library (a Python library for asynchronous programming) to complete two coding tasks.

Evaluation: Mastery was measured via a 14-question post-task quiz covering three domains: Debugging, Code Reading, and Conceptual Understanding.

Data
Quantitative: Task completion times, quiz scores (graded out of 27 points), and Python familiarity (calibrated via a warm-up task).

Qualitative/Behavioral: Keystroke logs and full transcripts of AI chat interactions to categorize user behavior into six distinct patterns.

Self-Reported: Participant surveys on task difficulty, enjoyment, and perceived learning.

Findings
Learning Impairment: The AI group scored significantly lower on the mastery quiz (17% lower average) than the control group, with the largest gap found in debugging skills.

Inconsistent Productivity: AI did not provide a statistically significant speed boost on average. While a subset of "delegators" finished faster, others spent significant time refining prompts or reading explanations.

Cognitive Engagement Matters: Skill formation depended on how the AI was used.

Low Scorers: Delegated entire blocks of code without asking for explanations.

High Scorers: Used AI to ask conceptual questions or requested line-by-line breakdowns of generated code.

The "Safety" Gap: The erosion of debugging skills suggests that developers who rely on AI to learn may eventually lack the expertise required to supervise or verify AI-generated code in high-stakes environments.

Limitations
Short Duration: The study measured "micro-learning" over a 75-minute window rather than longitudinal skill development over months.

Interface Constraints: The study used a chat-based interface; the researchers note that "agentic" tools (which require even less human input) might cause even greater skill erosion.

Incentive Alignment: As a lab study with paid crowd-workers, participants may have lacked the long-term professional incentive to master the library compared to a full-time employee.

Lack of Human Comparison: The study did not compare AI assistance against human mentorship or pair programming.

Here's the scoring for **Gemini 3 Fast — Paper Triage — Prompt A**, evaluated with your **Phase 1 rubric** (baseline prompt, lighter constraints).

Groundedness_Score: 2
Citation_Correctness_Score: 2
Usefulness_Score: 3
Failure_Tags: Interpretive_Overreach; Ungrounded_Claims; Missing_Citations

Notes:

- The summary captures the paper's general theme and experimental setup, but repeatedly **extends beyond what is strictly supported** by the paper, especially in interpretive framing such as "cost of convenience," "safety gap," and implications about high-stakes environments.
- Several details are presented with strong causal or normative language (e.g., erosion of debugging skills implying future supervision failure) that are **not directly measured or claimed** in the paper, reducing groundedness.
- No citations are provided, which limits citation correctness even under Prompt A.
- Despite these issues, the output is readable and moderately useful for orientation, though it blends empirical findings with editorial interpretation more than a neutral triage summary should.

model: Gemini 3 Fast
method: Paper Triage
prompt: B

llm output:

Contribution
The paper investigates the impact of AI assistance on skill formation and knowledge acquisition in software engineering. It specifically addresses how the use of AI tools during tasks requiring new concepts affects a developer's subsequent ability to understand, read, and debug code independently.

Method
The researchers conducted a between-subjects randomized experiment involving 52 participants (26 in a control group, 26 in a treatment group).

Task: Participants used the Python Trio library (specifically chosen for its unfamiliarity and "structured concurrency" concepts) to complete two coding tasks.

Conditions: The treatment group had access to a GPT-4o-based AI assistant, while the control group used only standard web searches and instructions.

Evaluation: Mastery was measured via a post-task 14-question quiz (27 points total) covering conceptual understanding, code reading, and debugging.

Data Collection: Researchers collected keystroke logs, AI interaction transcripts, and pre/post-task surveys.

Data
Participants: 52 professional or freelance programmers with 1+ years of Python experience but no prior Trio library experience.

Quantitative Metrics: Task completion times, quiz scores (standardized via item response theory), and self-reported learning/difficulty ratings on a 7-point scale.

Qualitative Metrics: Analysis of six distinct AI interaction patterns ranging from full delegation to conceptual questioning.

Findings
Skill Impairment: AI use significantly reduced skill formation, resulting in a 17% lower average quiz score (Cohen's d=0.738) compared to the control group.

Specific Deficits: The largest performance gap occurred in debugging questions, as the control group gained more experience troubleshooting errors manually.

Productivity Paradox: AI assistance did not provide significant average efficiency gains in completion time, though it did ensure all treatment participants finished the task (unlike the control group).

Interaction Matters: High quiz scores were maintained by participants who used AI for conceptual explanations or

cognitively engaged with the generated code. Low scores were tied to "full delegation" patterns.

Perception: AI users found the tasks easier but reported lower self-rated learning than the control group.

Limitations
Task Selection: The study used a single library and a chat-based interface; it did not test more automated "agentic" AI tools.

Task Length: Skill formation was measured over a one-hour window rather than a longitudinal period (months or years).

Participant Incentives: Being a paid study with crowd-workers, participants lacked the long-term incentive to master the library required in a real job setting.

Evaluation Scope: The study relied on a quiz rather than a secondary coding task or "human-assistance" counterfactual for evaluation.

Prompting Measurement: The study relied on self-reported AI familiarity rather than an objective measure of prompting fluency.

Here's the scoring for **Gemini 3 Fast — Paper Triage — Prompt B**, applying your **Phase 1 rubric** with **strict Prompt B guardrail enforcement**.

Groundedness_Score: 2
Citation_Correctness_Score: 2
Usefulness_Score: 3
Failure_Tags: Guardrail_Violation; Ungrounded_Detail; Overprecision; Missing_NotSpecified

Notes:

- While the response follows the required five-field structure, it repeatedly **introduces details that are not clearly specified in the paper** (e.g., item response theory scoring, ensuring all treatment participants finished, specific psychometric framing), which violates Prompt B's "do not infer" constraint.
- Several quantitative and methodological claims are presented with **high precision** (effect sizes, scoring methods, interaction taxonomy framing) without explicitly marking uncertainty or using "Not specified in the paper."
- No citations are provided, and claims that would require explicit grounding are stated as facts, lowering citation correctness.
- The output remains readable and informative, but compared to ChatGPT 5.2 Auto under Prompt B, it shows weaker compliance with guardrails and more speculative elaboration.

Prompt: A
method: Claim Evidence Extraction
Model: ChatGPT 5.2 Auto
Paper: The Good and Bad of AI Tools in Novice Programming Education
Output:

Here are five key claims about the impact of AI tools on software engineering (specifically novice programming education) extracted directly from the paper, each with a supporting snippet and a clear citation.
-
Key Claims from The Good and Bad of AI Tools in Novice Programming Education
#
Claim
Supporting Quote / Snippet
Citation
1
AI tool familiarity among novice programmers increases dramatically over time.

"The findings revealed a significant increase in AI tool familiarity (from 28% to 100%) and usage among students."
Zviel-Girshin, 2024
2
Student satisfaction with AI tools improves as they gain experience using them.
"Students' satisfaction with AI tools improved over time."
Zviel-Girshin, 2024
3
Novice programmers primarily use AI tools for low-level support tasks rather than full code generation.
"The most prevalent tasks for which novice programmers used AI tools included creating comments (91.7%), identifying and correcting bugs (80.2%), and seeking information (68.5%)."
Zviel-Girshin, 2024
4
AI tools provide meaningful learning benefits, including improved motivation and real-world relevance.
"AI tools significantly enhanced participants' learning experiences by assisting with information retrieval, bug identification, and writing comments… fostering essential literacies and skills that are crucial for understanding technology and preparing them for the future."
Zviel-Girshin, 2024
5
Over-reliance on AI tools can harm conceptual understanding and independent problem-solving skills.
"The risk of students developing an excessive or even blind reliance on these tools… can lead to a superficial understanding of programming concepts and a diminished ability to solve problems independently."
Zviel-Girshin, 2024

Here's the scoring for **ChatGPT 5.2 Auto — Claim Evidence Extraction — Prompt A**, evaluated with your **Phase 1 rubric**.

Groundedness_Score: 2

Citation_Correctness_Score: 2

Usefulness_Score: 3

Failure_Tags: Citation_Format_Error; Overgeneralized_Claims; Evidence_Paraphrase_Drift

Notes:

- While the claims broadly reflect themes present in the paper, several are **overgeneralized** (e.g., "provide meaningful learning benefits") and move beyond what is tightly supported by the quoted snippets.
- The "supporting quotes" are often **loosely paraphrased summaries** rather than verbatim or closely bounded excerpts, weakening the claim–evidence linkage required even under Prompt A.
- Citations are provided only as author/year and **do not clearly reference specific sections or evidence locations**, reducing citation correctness.
- The table is readable and moderately useful for orientation, but it lacks the precision and traceability needed for high-confidence claim–evidence extraction.

Prompt: B
Model: ChatGPT 5.2 Auto
method: Claim Evidence Extraction
Paper: The Good and Bad of AI Tools in Novice Programming Education
Output:

Claim
Evidence Snippet
Citation (source_id, chunk_id)
Novice programmers' familiarity with AI tools increased substantially over the course.
"Initially, only 28% of the teams reported feeling familiar

with the usage of AI tools. However, by the end of the course, this familiarity had increased to 100%."
(GoodBadAIProgrammingEd, Results-3.1)
Students' usage of AI tools increased over time, even when not required.
"Cohran's Q = 34.839, $p < 0.001$, meaning that there were significant differences... indicating that students used more AI tools."
(GoodBadAIProgrammingEd, Results-3.2)
Students' satisfaction with AI tool results improved over time.
"The differences in medians were significant... χ^2 (2) = 11.594, p = 0.003, meaning that the answers were better."
(GoodBadAIProgrammingEd, Results-3.3)
The most common AI-assisted tasks for novice programmers were commenting, bug fixing, and information seeking.
"Creation of comments... 91.7%, bugs identification and correction—80.2%, information seeking—68.5%."
(GoodBadAIProgrammingEd, Results-3.4)
Students expressed concerns about over-reliance on AI tools and inaccurate outputs.
"Common concern categories... inaccurate or misleading information... 89.0%, over-reliance on AI tools... 75.3%."
(GoodBadAIProgrammingEd, Results-3.5)

Here's the scoring for **ChatGPT 5.2 Auto — Claim Evidence Extraction — Prompt B**, evaluated with your **Phase 1 rubric** and **strict Prompt B guardrails**.

Groundedness_Score: 4
Citation_Correctness_Score: 3
Usefulness_Score: 4
Failure_Tags: Minor_Citation_Format_Ambiguity

Notes:

- Each claim is **directly supported by a clearly matching evidence snippet**, and the snippets are either verbatim or tightly bounded paraphrases, satisfying Prompt B's strongest requirement.
- The model avoids speculation and does not introduce claims beyond what the cited results sections support, showing strong grounding and good uncertainty discipline.
- Citation format follows the required `(source_id, chunk_id)` structure, though the chunk identifiers (e.g., `Results-3.1`) are **section-level rather than true chunk IDs**, creating slight ambiguity but not fabrication.
- Overall, this is a high-quality claim–evidence extraction that demonstrates the benefit of Prompt B guardrails in reducing hallucination and improving traceability.

Prompt: A Model: Gemini 3 Fast method: Claim Evidence Extraction Paper: The Good and Bad of AI Tools in Novice Programming Education Output:

Based on the complete text of the paper, here are five key claims regarding the impact of AI tools on software engineering and programming education:ClaimSupporting QuoteCitationAI tools improve with user experience through better prompt engineering."The differences in medians were significant... meaning that the answers were better. However, since students used the same language model, this result suggests that they improved their prompt-engineering skills over time."Zviel-Girshin (2024), Section 3.3AI tools are highly effective for "drudge work" like documentation."The most prevalent tasks for which novice programmers used AI tools included creating comments (91.7%)... The efficiency of AI tools also led to the over-automation of some programming tasks, such as generating comments, with 92% of teams fully outsourcing this task to AI."Zviel-Girshin (2024), Sections 3.4 & 4.3AI provides immediate feedback that increases developer motivation."Additionally, some participants noted that receiving feedback from AI tools

when they were stuck boosted their self-efficacy and motivation to learn."Zviel-Girshin (2024), Section 4.2AI-generated content often contains "believable" but incorrect logic."Common concern categories... included inaccurate or misleading information/answers... [and] incorrect explanations which appear believable to novices."Zviel-Girshin (2024), Section 3.5Excessive AI use can turn developers into passive recipients rather than active problem solvers."AI is often doing most of the work. This dynamic can erode the learning experience, as students become passive recipients rather than active participants in the problem-solving process."Zviel-Girshin (2024), Section 4.3

Here's the scoring for **Gemini 3 Fast — Claim Evidence Extraction — Prompt A**, evaluated with your **Phase 1 rubric** (baseline prompt, lighter constraints).

Groundedness_Score: 2
Citation_Correctness_Score: 3
Usefulness_Score: 3
Failure_Tags: Overgeneralized_Claims; Interpretive_Paraphrase; Claim_Evidence_Drift

Notes:

- Several claims are **interpretive extensions** of the cited text rather than strictly supported statements (e.g., "improve with user experience through better prompt engineering," "turn developers into passive recipients"), which weakens groundedness even though they are thematically aligned with the paper.
- Evidence snippets are sometimes **selectively quoted and rhetorically reframed**, making the claim stronger or more generalized than what the text directly supports.
- Citations reference plausible sections of the paper and are more specific than simple author–year citations, resulting in a stronger citation correctness score than some other Prompt A outputs.
- Overall, the table is readable and moderately useful, but it blends empirical findings with analytical interpretation, limiting reliability for strict claim–evidence extraction.

Prompt: B Model: Gemini 3 Fast method: Claim Evidence Extraction Paper: The Good and Bad of AI Tools in Novice Programming Education Output:

Claim,Evidence Snippet,"Citation (source_id, chunk_id)"
The most frequent task for which novice students used AI was generating code comments.,"""Categories that emerged from the analysis... creation of comments mentioned by 67 teams (91.7%)""","(The Good and Bad of AI Tools in Novice Programming Education, 3.4)"
Students' satisfaction with AI tool results improved significantly over the 12-week course.,"""The Friedman test... comparing the responses to the item, 'I was happy with the results provided by AI tools'... The differences in medians were significant... meaning that the answers were better.""","(The Good and Bad of AI Tools in Novice Programming Education, 3.3)"
The primary concern among students regarding AI tools is the risk of receiving inaccurate information.,"""Common concern categories... inaccurate or misleading information/answers mentioned by 65 teams (89.0%)""","(The Good and Bad of AI Tools in Novice Programming Education, 3.5)"
Novice programmers improved their prompt-engineering skills as they progressed through the semester.,"""Since students used the same language model, this result suggests that they improved their prompt-engineering skills over time.""","(The Good and Bad of AI Tools in Novice Programming Education, 3.3)"
AI tools provided a motivational boost to students when they encountered obstacles in coding.,"""receiving feedback from AI tools when they were stuck boosted their self-efficacy and motivation to learn.""","(The Good and Bad of AI Tools in Novice Programming Education, 4.2)"

Here's the scoring for **Gemini 3 Fast — Claim Evidence Extraction — Prompt B**, evaluated with your **Phase 1 rubric** and **strict Prompt B guardrails**.

Groundedness_Score: 3
Citation_Correctness_Score: 2
Usefulness_Score: 3
Failure_Tags: Citation_Format_Error; Guardrail_Violation; Overinterpretation

Notes:

- Most claims are reasonably supported by the provided evidence snippets, and the snippets are largely verbatim or closely paraphrased, which supports solid groundedness overall.
- However, several claims subtly **over-interpret the evidence** (e.g., framing improved satisfaction or inferred prompt-engineering skill as broad developmental outcomes rather than context-specific observations), which prevents a top groundedness score.
- Citation format does **not comply with Prompt B's required** `(source_id, chunk_id)` **schema**: section numbers are used instead of chunk IDs, constituting a guardrail violation and lowering citation correctness.
- The output is readable and informative, but its methodological traceability and strict compliance are weaker than the ChatGPT 5.2 Auto Prompt B counterpart.

Prompt: A
method: Claim Evidence Extraction
Model: ChatGPT 5.2 Auto
Paper: Measuring GitHub Copilot's Impact on Productivity
Output:

Here are five key claims drawn directly from the paper, each supported by a quote from the text and a clear citation.

Claim

Supporting Quote / Snippet

Citation

Acceptance rate of Copilot suggestions is the strongest predictor of developers' perceived productivity.

"We find acceptance rate (accepted _ per _ shown) most positively predicts users' perception of productivity."

(Measuring GitHub Copilot's Impact on Productivity, What Drives Perceived Productivity)

Acceptance rate correlates better with perceived productivity than more detailed contribution or persistence metrics.

"We were surprised to find acceptance rate … to be better correlated with reported productivity than our measures of persistence."

(Measuring GitHub Copilot's Impact on Productivity, Conclusions)

More detailed persistence-based metrics show weaker correlations with perceived productivity than acceptance rate.

"All persistence measures were less well correlated than acceptance rate."

(Measuring GitHub Copilot's Impact on Productivity, What Drives Perceived Productivity)

Junior developers report higher productivity gains from Copilot and accept suggestions more frequently than senior developers.

"Junior developers not only report higher productivity gains; they also tend to accept more suggestions."

(Measuring GitHub Copilot's Impact on Productivity, Experience)

Experienced developers benefit less in code quality improvements but still gain productivity through flow and reduced repetitive work.

"Experienced developers who are already highly skilled are less likely to write better code with Copilot, but Copilot can assist their productivity in other ways,

particularly when engaging with new areas and automating routine work."
(Measuring GitHub Copilot's Impact on Productivity, Experience)

Here's the scoring for **ChatGPT 5.2 Auto — Claim Evidence Extraction — Prompt A**, evaluated with your **Phase 1 rubric** (baseline prompt, lighter constraints).

Groundedness_Score: 3
Citation_Correctness_Score: 3
Usefulness_Score: 4
Failure_Tags: Minor_Claim_Redundancy; Section_Level_Citations

Notes:

- The claims are well aligned with the paper's reported analyses and are generally **well supported by closely matching evidence snippets**, showing solid grounding overall.
- There is some **redundancy** across the first three claims (all centered on acceptance rate vs. persistence metrics), which slightly limits conceptual breadth but does not undermine factual accuracy.
- Citations correctly reference relevant paper sections, though they are **section-level rather than granular evidence locations**, which caps citation correctness short of a perfect score.
- Overall, this is a clear, faithful, and highly useful extraction that would be valuable for understanding the paper's core empirical insights, especially regarding perceived productivity drivers.

Method: Claim Evidence Extraction
Prompt: B
Model: ChatGPT 5.2 Auto
Paper: Measuring GitHub Copilot's Impact on

Productivity
Output:

Claim
Evidence Snippet
Citation (source_id, chunk_id)
Acceptance rate is the strongest usage-based predictor of developers' perceived productivity when using GitHub Copilot.
"We find acceptance rate (accepted _ per _ shown) most positively predicts users' perception of productivity… Of all usage measurements, acceptance rate correlates best with aggregate productivity ($\rho = 0.24$, $P < 0.0001$)."
(Measuring GitHub Copilot's Impact on Productivity, What Drives Perceived Productivity)
Acceptance rate correlates better with perceived productivity than more detailed contribution or persistence measures.
"Our results support the principle that acceptance rate can be used for coarse-grained monitoring… This ratio of shown suggestions being accepted correlates better than more detailed measures of contribution."
(Measuring GitHub Copilot's Impact on Productivity, Results)
Persistence-based metrics of accepted code are less correlated with perceived productivity than acceptance rate.
"All persistence measures were less well correlated than acceptance rate."
(Measuring GitHub Copilot's Impact on Productivity, What Drives Perceived Productivity)
Junior developers report higher productivity gains and accept more suggestions than more experienced developers.
"Junior developers not only report higher productivity gains; they also tend to accept more suggestions."
(Measuring GitHub Copilot's Impact on Productivity, Experience)

Acceptance rate varies significantly by time of day and day of week, with higher rates outside typical working hours.
"We observe strong regular patterns in overall acceptance rate… weekends… average acceptance rate… 23.5%… typical working hours… much lower at 21.2%." (Measuring GitHub Copilot's Impact on Productivity, Variation over Time)

Here's the scoring for **ChatGPT 5.2 Auto — Claim Evidence Extraction — Prompt B**, using your **Phase 1 rubric** with **strict Prompt B guardrails**.

Groundedness_Score: 4
Citation_Correctness_Score: 3
Usefulness_Score: 4
Failure_Tags: Minor_Citation_Format_Ambiguity; Conceptual_Redundancy

Notes:

- All five claims are **directly supported by tightly matching evidence snippets**, with several quotes including precise statistics (e.g., correlations and p-values), demonstrating excellent grounding.
- The model avoids speculation and stays within the scope of reported results, fully complying with Prompt B's "no inference" constraint.
- Citation format follows the `(source_id, chunk_id)` structure, but chunk identifiers are **section-level labels rather than true chunk IDs**, creating minor ambiguity and preventing a perfect citation score.
- There is some conceptual overlap among acceptance-rate–focused claims, but the set remains highly useful for understanding the paper's core empirical findings.

Method: Claim Evidence Extraction
Prompt: A

Model: Gemini 3 Fast
Paper: Measuring GitHub Copilot’s Impact on Productivity
Output:

Based on the paper provided, here are five key claims regarding the impact of AI tools (specifically GitHub Copilot) on software engineering:

Key Claims: AI Tools in Software Engineering

Key Claim	Supporting Quote/Snippet	Citation
Acceptance rate is the best predictor of productivity.	"We find that acceptance rate of shown suggestions is a better predictor of perceived productivity than the alternative measures."	"Measuring GitHub Copilot’s Impact on Productivity" (Results section)
Junior developers experience higher gains from AI tools.	"Junior developers not only report higher productivity gains; they also tend to accept more suggestions."	"Measuring GitHub Copilot’s Impact on Productivity" (Experience section)
Experienced developers use AI for routine work rather than code quality.	"...experienced developers who are already highly skilled are less likely to write better code with Copilot, but Copilot can assist their productivity in other ways, particularly when engaging with new areas and automating routine work."	"Measuring GitHub Copilot’s Impact on Productivity" (Experience section)
Acceptance rate varies by the time of day and week: acceptance rate is much lower at 21.2% [compared to the weekend and non-working hours]."		"Measuring GitHub Copilot’s Impact on Productivity" (Variation over Time section)
The value of AI lies in goal progress rather than perfect		

ChatGPT 5.2

accuracy.
"GitHub Copilot's central value lies not in being the way users enter most of their code. Instead, it lies in helping users to make the best progress toward their goals."
"Measuring GitHub Copilot's Impact on Productivity" (Conclusions section)

Here's the scoring for **Gemini 3 Fast — Claim Evidence Extraction — Prompt A**, using your **Phase 1 rubric** (baseline prompt, lighter constraints).

Groundedness_Score: 3
Citation_Correctness_Score: 3
Usefulness_Score: 3
Failure_Tags: Minor_Overgeneralization; Section_Level_Citations

Notes:

- The claims are generally **well aligned with the paper's findings**, and most supporting snippets closely match the stated claims, indicating solid grounding overall.
- A few claims are **slightly abstracted or interpretive** (e.g., "the value of AI lies in goal progress rather than perfect accuracy"), which is thematically accurate but framed more normatively than the underlying evidence.
- Citations correctly reference relevant sections of the paper, but they remain **section-level rather than precise evidence locations**, which limits citation correctness short of a top score.
- The output is clear and readable, but compared to ChatGPT 5.2 Auto under Prompt B, it offers less precision and traceability.

Method: Claim Evidence Extraction
Prompt: B

Model: Gemini 3 Fast
Paper: Measuring GitHub Copilot’s Impact on Productivity
Output:

To understand the impact of AI-assisted software engineering, researchers analyzed how developer interactions with GitHub Copilot relate to their own feelings of productivity.
Extracted Claims and Evidence
Claim
Evidence Snippet
Citation (source_id, chunk_id)
Acceptance rate is the strongest predictor of a developer's perceived productivity among usage metrics.
"We find that acceptance rate of shown suggestions is a better predictor of perceived productivity than the alternative measures."
(Paper, 1)
Offline evaluation of code completion systems often fails to reflect real-world performance accurately.
"This metric fell from 90% in offline evaluation to 70% when used online."
(Paper, 1)
Highly experienced developers may derive less improvement in code quality from Copilot than their junior peers.
"Both language proficiency and years of experience negatively predict developers agreeing that Copilot helps them write better code."
(Paper, 2)
Usage patterns for Copilot, specifically acceptance rates, fluctuate based on the time of day and week.
"Acceptance rate typically increases over the board... we distinguish three different time regimes... The weekend... typical non-working hours... typical working hours."
(Paper, 2)
Perceived productivity is best measured as a multidimensional concept rather than a single metric.

"The SPACE framework defines five dimensions of productivity: Satisfaction and well-being, Performance, Activity, Communication and collaboration, and Efficiency and flow."
(Paper, 2)

Here's the scoring for **Gemini 3 Fast — Claim Evidence Extraction — Prompt B**, evaluated with your **Phase 1 rubric** and **strict Prompt B guardrails**.

Groundedness_Score: 2
Citation_Correctness_Score: 1
Usefulness_Score: 2
Failure_Tags: Hallucinated_Claim; Citation_Format_Error; Guardrail_Violation; External_Knowledge_Leak

Notes:

- Several claims are **not directly supported by the provided evidence snippets** or drift beyond the paper's empirical findings (e.g., offline vs. online evaluation performance and the multidimensional productivity claim via the SPACE framework, which introduces external framing rather than extracted results).
- Citation formatting **does not comply** with Prompt B requirements: `(Paper, 1)` and `(Paper, 2)` are ambiguous and do not represent valid `source_id, chunk_id` pairs, resulting in a severe citation correctness penalty.
- The final claim relies on **external conceptual knowledge (SPACE framework)** rather than a claim grounded in the paper's reported results, violating the "do not rely on external knowledge" guardrail.
- Overall, the output shows weak compliance with Prompt B constraints and limited usefulness for rigorous claim–evidence extraction compared to the ChatGPT 5.2 Auto Prompt B outputs.