Lecture 3: Improving Performance

Context Engineering Techniques for Better Results

University of Chicago

January 25, 2026

Outline

Review: What We've Covered

Lecture 1: Foundations

- Basic AI terminology (LLMs, tokens, context windows, APIs)
- How to work with these models programmatically
- Understanding pricing and token economics

Lecture 2: Building AI Systems

- Vertical slices - prove end-to-end flow works first
- Crawl, Walk, Run - start simple, add complexity incrementally
- Breaking problems into clear inputs and outputs

Where We're Going

Last Time: We built a basic resume scoring system

- Single prompt: resume + job requirements → 0-100 score

Today: Learn techniques to improve AI system performance

- **Lecture**: Walk through a simple example (expense validation)
- **Work Session**: Apply these same techniques to improve the resume scorer

Outline

The Core Problem

It's difficult to know what does and does not work with AI Systems.

- Results are inconsistent and non-deterministic
- Hard to debug when things go wrong
- Model hallucinates or makes up information
- Edge cases get mishandled
- We (often) can't explain *why* the model decided something

This is why we use vertical slices – to identify processes that require improvement.

The Core Problem

Our resume scorer from Lecture 2 had issues:

- Scores ranged from 0-100, but were they *good*?
- Results tended to be bunchy (most scores clustered in narrow ranges)
- Difficult to interpret what a "65" meant vs. a "75"
- No clear explanation for why a candidate got their score
- Hard to defend decisions to hiring managers

How do we make this better?

- Break down what we're measuring
- Provide evidence for claims
- Make the scoring process more transparent

The Core Problem

Solution: Engineering the context more carefully

- How?
    - Break complex problems into simpler steps
    - Ground the model with evidence requirements
    - Provide examples to guide behavior

Outline

Expense Report Validator

Task: Validate employee expense reports for compliance

Input: Receipt/invoice text + description

Required Outputs:

- Expense type (Travel, Office Supplies, Meals/Entertainment)
- Date and location
- Compliance issues

Compliance Rules:

1. **Amount limits**: Different limits per category
2. **Required fields**: Travel requires trip purpose, Meals require attendees
3. **Prohibited expenses**: No alcohol reimbursement

Example 1: Travel Expense

Receipt:

```
Conference Badge
AI Summit 2024
Jan 15-17, 2024
San Francisco, CA

Registration:  $495.00
Processing Fee:  $15.00
------------------------
Total:  $510.00
```

Expected Output:

- **Type**: Travel
- **Date**: Jan 15-17, 2024
- **Location**: San Francisco, CA
- **Amount**: $510.00
- **Required fields**: Trip purpose (conference), destination
- **Compliance**: Pass

Example 2: Meals with Alcohol

Receipt:

```
The Steakhouse
Boston, MA
Jan 20, 2024

2x Ribeye Steak:  $68.00
1x Caesar Salad:  $14.00
1x House Wine:  $16.00
Tax:  $9.80
-------------------------
Total:  $107.80
```

Expected Output:

- **Type**: Meals/Entertainment
- **Date**: Jan 20, 2024
- **Location**: Boston, MA
- **Amount**: $107.80
- **Required fields**: Business purpose, attendees
- **Compliance**: Fail
- **Issue**: Contains prohibited alcohol ($16.00)

Initial Approach: Monolithic Prompt

Initial approach: One prompt does everything

Prompt

Analyze this expense report and return JSON with: expense_type, date, location, amount, compliance_issues, required_fields_missing, prohibited_items_found.
Receipt: [receipt text here]
Check all compliance rules...

This seems reasonable - what could go wrong?

Problem: Inconsistent and Unreliable

What goes wrong:

- Misclassifies expense types (calls a conference "Office Supplies" instead of "Travel")
- Misses conditional logic (doesn't ask for trip purpose on travel expenses)
- Sometimes flags issues that don't apply to the expense type
- Results vary when you run the same receipt twice

Why:

- The prompt is trying to do too many things at once
- Conditional logic ("if travel, then check X") is implicit
- Model has to keep track of multiple rules simultaneously

Outline

Solution: Decompose Into Stages

Better approach: Break into sequential steps with clear outputs

Step 1: Classify Type
Output: expense_type

↓

Step 2: Extract Fields
Based on type

↓

Step 3: Check Compliance
Type-specific rules

Key idea: Each step is simple and has a single responsibility

Step 1: Classify Expense Type

First prompt: Just classify the type

Classification Prompt

Classify this expense into one of three categories:

- Travel (flights, hotels, conferences, transportation)
- Office Supplies (equipment, software, books)
- Meals/Entertainment (restaurants, client dinners)

Receipt: [Dinner at restaurant, $48.50]
Return JSON: {"expense_type": "..."}

Result: `{"expense_type": "Meals/Entertainment"}`

Step 2: Conditional Logic in Code

Now we can apply type-specific logic in our code:

Python Logic

```
if expense_type == "Travel":
    # Extract: trip_purpose, destination, dates
    run_travel_prompt()

elif expense_type == "Meals/Entertainment":
    # Extract: attendees, business_purpose
    run_meals_ent_prompt()

elif expense_type == "Office Supplies":
    run_office_supplies_prompt()
```

Key insight: LLMs are great at classification; code is great at conditional logic

Results After Decomposition

What improved:

- Classification is more consistent and reliable
- Type-specific rules are explicit in code
- Easy to debug (we can inspect the output of each step)
- Can test classification independently from validation

Takeaway:

Simplify each prompt by breaking complex tasks into stages
Use LLMs for unstructured→structured transformation
Use code for logic and conditional rules

Outline

New Problem: Hallucination

Context: We're checking for prohibited expenses (alcohol)

Prompt: Does this receipt contain alcohol? Return JSON: {"contains_alcohol": true/false, "reason": "..."}
Receipt: [Restaurant bill, $48.50, two entrees and soft drinks]

Result: `{"contains_alcohol": true, "reason": "Likely includes wine or beer with dinner"}`

Problem: The model *assumed* there was alcohol when the receipt never mentioned it!

Why This Happens

LLMs are pattern matchers, not database lookups:

- Restaurant + dinner → often includes alcohol in training data
- Model generates "plausible" answers based on patterns
- Without explicit evidence requirement, it fills in gaps

This is dangerous:

- False rejections (denying valid expenses)
- Compliance issues (wrong reasons in audit trail)
- Loss of trust in the system

Solution: Require Citations

Better prompt: Demand evidence from the receipt or ask to verify before returning (which can add $$$)

Grounded Prompt

Does this receipt contain alcohol?
You MUST cite exact quotes from the receipt to support your answer.
If alcohol is present, provide the exact line items that mention it.
Receipt: [Restaurant bill, $48.50, two entrees and soft drinks]
Return JSON:

```
{
  "contains_alcohol": true/false,
  "evidence": ["exact quote 1", "exact quote 2"],
  "reason": "explanation based on evidence"
}
```

Results After Grounding

New result:

```
{
  "contains_alcohol": false,
  "evidence": [],
  "reason": "No alcohol items found in receipt.
            Items listed: 2x Lunch Special, 2x Soft Drink"
}
```

What improved:

- Model is forced to ground claims in actual receipt text
- Hallucinations are reduced significantly
- We get an audit trail (can verify the evidence ourselves)
- False positives drop dramatically

Grounding Best Practices

How to apply grounding:

1. Require exact quotes or citations for any factual claims
2. Ask for evidence *before* conclusions
3. Validate that evidence exists in the source material
4. Use structured output with separate "evidence" field

When to use it:

- Compliance/regulatory contexts (must justify decisions)
- Extracting facts from documents
- Any scenario where hallucination is risky

Citations turn the LLM from a "creative writer" into an "evidence-based analyst"

Outline

Another Problem: Edge Cases

Context: Classification works well for obvious cases

But what about ambiguous expenses?

- Conference registration: Travel or Office Supplies?
- Team building dinner: Meals or Office Supplies?
- Online course subscription: Office Supplies or ???
- Software for a specific project: Office Supplies or ???

Problem: Model makes inconsistent decisions on edge cases

- Conference registration sometimes classified as "Office Supplies"
- No clear policy guidance in the prompt

Solution: Provide Examples

Better prompt: Show the model what good classification looks like

Few-Shot Prompt

Classify this expense. Here are some examples:
Example 1: Receipt: Conference registration, AI Summit 2024, $500 Classification: Travel (conferences are travel-related)
Example 2: Receipt: Office chair from Staples, $200 Classification: Office Supplies (furniture and equipment)
Example 3: Receipt: Team dinner at local restaurant, $120 Classification: Meals/Entertainment (company-sponsored meals)
Now classify: Receipt: [Your receipt here]

Types of Examples to Include

Good few-shot examples include:

1. **Positive examples**: Correct classifications
2. **Edge cases**: Ambiguous situations with your preferred handling
3. **Negative examples**: Common mistakes to avoid

Example - Negative case: Receipt: Laptop purchased for work, $1,200
Incorrect: Meals/Entertainment
Correct: Office Supplies (technology equipment)
Reason: Even though expensive, hardware is office supplies, not travel

How many examples? 2-5 is usually enough; more isn't always better

Results After Few-Shot Examples

What improved:

- Edge cases are now handled consistently
- Model learns your organization's specific policies
- Classification accuracy improves on ambiguous cases
- Fewer surprises and unexpected categorizations

Tradeoff:

- More tokens in prompt (costs more, takes longer)
- Need to maintain example set
- But: much more reliable results

Examples teach the model your specific policies and edge case handling

Outline

Additional Performance Techniques

Beyond decomposition, grounding, and examples:

1. **Role/Persona Specification**
   - "You are a careful compliance officer reviewing expenses..."
   - Sets tone and behavior expectations
2. **Providing Domain Context**
   - Include relevant company policies in the prompt
   - Specific rules and thresholds from your domain
3. **Adding Guardrails**
   - Structured output schemas (JSON with required fields)
   - Validation logic to catch impossible outputs
   - Allowed/blocked lists (e.g., only these 3 expense types)

More Advanced Techniques

4. **Chain-of-Thought Prompting**
    - Ask model to "show your reasoning step by step"
    - Improves accuracy on complex logic problems
    - Makes debugging easier
5. **Temperature Tuning**
    - Lower temperature (0.0-0.3) for consistency
    - Higher temperature (0.7-1.0) for creativity
    - Classification and extraction: use low temperature
6. **Output Validation**
    - Check LLM outputs with deterministic code
    - Re-prompt if validation fails
    - Combine AI flexibility with code reliability

The Three Core Techniques (Summary)

1. Decompose

Break complex prompts into stages

Use code for conditional logic

2. Ground

Require citations

Demand evidence

Reduce hallucination

3. Examples

Few-shot prompting

Show edge cases

Guide behavior

Use these in combination for best results

Outline

Today's Workflow

You will work with 3 randomly sampled resumes:

1. **Baseline**: Run the monolithic scorer
   - Single prompt: resume $\rightarrow$ 0-100 score
   - Collect results in a DataFrame
2. **TODO 1**: Extract years of experience
   - Write a focused prompt to extract only years of experience
   - Require citations/evidence from resume
   - Run on all 3 samples, create DataFrame
3. **TODO 2**: Extract and score technologies
   - Extract technologies/skills from resume
   - Compare against required technologies in job req
   - Score 0-100 based on match percentage

Today's Workflow (continued)

4. **TODO 3**: Combine and compare
   - Merge year data with technology scores
   - Create a composite score from extracted features
   - Compare against monolithic 0-100 scores
   - Analyze: Which is more consistent? Which explains better?
5. **TODO 4**: Extensions (pick one or more)
   - Education requirements extraction
   - Leadership/mentoring experience
   - Scale to 20 resumes and compare costs

Key Takeaways for Your Work

Apply the three core techniques:

- **Decomposition**: Break scoring into feature extraction steps
- **Grounding**: Require citations for years of experience
- **Examples**: (Optional) Add few-shot examples for edge cases

What to look for:

- Are decomposed scores more explainable than monolithic scores?
- Can you justify each score with specific extracted features?
- How do token costs compare between approaches?
- Which approach would you trust more in production?

Remember: The goal is *transparency* and *consistency*, not just higher scores!

Let's Get Started!

Open the notebook:

`lecture_3_resume_scorer_improvement.ipynb`

Work through TODOs 1-4

Compare your results

Discuss with your team