Student Task Breakdown — 2-Week Sprint

Scope

Duration: Feb 17 – Feb 28 (2 weeks)
Budget: 8 hrs/week per student = 16 hrs each, 48 hrs total
Goal: Working backend API + clickable Figma prototype. By the end of this sprint, the frontend and backend can start talking to each other.

Supervision cadence

- **Monday AM:** Send task briefs (copy from this doc)
- **Wednesday:** Async check-in — each student posts a screenshot or code snippet of where they are
- **Friday:** Deliverable review — 15 min call or async demo

Mario & Steven — Backend (Python)

Week 1: Foundation (Feb 17–21)

Goal: Get TransformerLens running, understand the model, build the first API endpoint.

Task 1.1: Environment Setup (~2 hrs)

Who: Steven, Mario

What to do:

- Create a project folder `interpretative-interfaces-backend/`
- Set up a Python virtual environment (`python -m venv venv`)
- Install dependencies: `pip install transformer_lens flask numpy scikit-learn`
- Create `requirements.txt` with pinned versions
- Load GPT-2 small and run one forward pass to confirm it works:

```python
from transformer_lens import HookedTransformer
model = HookedTransformer.from_pretrained("gpt2-small")
logits, cache = model.run_with_cache("Hello world")
print(logits.shape)  # should be (1, 2, 50257)
print(cache["resid_post", 0].shape)  # should be (1, 2, 768)
```

Done when: you can run the above snippet on your laptops without errors.

Resources:

- TransformerLens install: https://github.com/TransformerLensOrg/TransformerLens
- Python venv docs: https://docs.python.org/3/library/venv.html

---

Task 1.2: Work Through the "50 Lines" Tutorial (~3 hrs)

Who: Steven, Mario

What to do:

- Open the tutorial notebook in Google Colab (free, no GPU needed for GPT-2 small)
- Run every cell. Don't just read ; type the code, look at the outputs
- For each section, write a 1-sentence comment explaining what it does
- **Key things to understand:**
  - `model.run_with_cache()` → how to get all activations
  - `cache["resid_post", layer]` → residual stream at a layer (token embeddings)
  - `cache["blocks.X.attn.hook_pattern"]` → attention weights
  - `model.to_str_tokens()` → tokenization
- Save your annotated notebook as `notebooks/tutorial-walkthrough.ipynb`

Done when: When you can explain in your own words: (1) what the residual stream is, (2) how to get a token's embedding at layer 5, (3) what attention patterns show.

Resources:

- Tutorial: How-to Transformer Mechanistic Interpretability in 50 Lines
- TransformerLens demo notebook: https://transformerlensorg.github.io/TransformerLens/content/getting_started_mech_interp.html

---

Task 1.3: Tokenization Function + Notebook (1.5 hrs)

Who: Steven

What to do:

- Create `notebooks/tokenization.ipynb`
- Write a function that takes a string and returns a list of token objects:

```
def tokenize(text: str) -> list[dict]:
    """Returns list of {index, token_str, token_id} for each token."""
    tokens = model.to_tokens(text)          # tensor of token IDs
    str_tokens = model.to_str_tokens(text)  # list of strings
    return [
        {"index": i, "token_str": s, "token_id": int(tokens[0, i])}
        for i, s in enumerate(str_tokens)
    ]
```

- Test with 3 different inputs (short sentence, longer sentence, sentence with unusual words)
- Note any surprises (subword tokens, spacing, special tokens like BOS)

Done when: Function works, tested with 3 inputs, outputs saved in the notebook.

Task 1.4: Embedding Extraction Across All Layers (~2 hrs)

Who: Mario

What to do:

- Create `notebooks/embedding_extraction.ipynb`
- Write a function that takes text and a token index, and returns that token's embedding at every layer:

```
def get_token_trajectory(text: str, token_index: int) -> list[dict]:
    """Returns the residual stream vector for one token at each layer."""
    _, cache = model.run_with_cache(text)
    n_layers = model.cfg.n_layers  # 12 for GPT-2 small
    trajectory = []
    for layer in range(n_layers):
        embedding = cache["resid_post", layer][0, token_index, :]  # shape
(768,)
        trajectory.append({
```

```
            "layer": layer,
            "embedding": embedding.tolist()  # convert tensor to list
        })
    return trajectory
```

- Test: extract trajectory for the word "cat" in "The cat sat on the mat"
- Print the shape and first 5 values at each layer to verify they change

Done when: Function works, trajectory returned for all 12 layers, values visibly differ across layers.

Task 1.5: Dimensionality Reduction (~2.5 hrs)

Who: Steven

What to do:

- Create `notebooks/dimensionality_reduction.ipynb`
- Write a function that takes multiple token trajectories and reduces them to 2D using PCA:

```
from sklearn.decomposition import PCA
import numpy as np

def reduce_trajectories(trajectories: dict[int, list]) -> dict[int,
list[dict]]:
    """
    Input: {token_index: [768-dim embedding per layer]}
    Output: {token_index: [{layer, x, y} per layer]}

    All tokens share the same PCA space so trajectories are comparable.
    """
    # Collect ALL embeddings into one matrix for joint PCA
    all_embeddings = []
    labels = []  # (token_index, layer) pairs
    for token_idx, layers in trajectories.items():
        for layer_data in layers:
            all_embeddings.append(layer_data["embedding"])
            labels.append((token_idx, layer_data["layer"]))

    all_embeddings = np.array(all_embeddings)
    pca = PCA(n_components=2)
    coords_2d = pca.fit_transform(all_embeddings)
```

```python
    # Reconstruct per-token trajectories with 2D coordinates
    result = {}
    for i, (token_idx, layer) in enumerate(labels):
        if token_idx not in result:
            result[token_idx] = []
        result[token_idx].append({
            "layer": layer,
            "x": float(coords_2d[i, 0]),
            "y": float(coords_2d[i, 1])
        })
    return result
```

- Test: reduce 2-3 tokens from the same sentence, plot the trajectories with matplotlib
- **Bonus:** Try UMAP (`pip install umap-learn`) as an alternative and compare

Done when: 2D scatter plot showing 2-3 token trajectories through 12 layers, each trajectory is a connected path of dots. Save the plot in the notebook.

Resources:

- scikit-learn PCA: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
- UMAP docs: https://umap-learn.readthedocs.io/en/latest/

Task 1.6: Flask App Skeleton + `/tokenize` Endpoint (~2 hrs)

Who: Mario

What to do:

- Create `app.py` with a basic Flask server
- Move the tokenization function from the notebook into `model_utils.py`
- Load the model once at startup (global variable — it's fine for a prototype)
- Implement the first endpoint:

```
POST /tokenize
Body: {"text": "The cat sat on the mat"}
Response: {
  "tokens": [
    {"index": 0, "token_str": "The", "token_id": 464},
    {"index": 1, "token_str": " cat", "token_id": 3797},
    ...
```

```
  ]
}
```

- Add CORS headers (`pip install flask-cors`) so a browser frontend can call it
- Test with `curl`:

  ```
  curl -X POST http://localhost:5000/tokenize \
    -H "Content-Type: application/json" \
    -d '{"text": "The cat sat on the mat"}'
  ```

Done when: `curl` returns correct JSON. Server starts without errors.

Resources:

- Flask quickstart: https://flask.palletsprojects.com/en/stable/quickstart/
- flask-cors: https://flask-cors.readthedocs.io/en/latest/

Week 1 File Structure

```
interpretative-interfaces-backend/
├── venv/
├── requirements.txt
├── app.py                          # Flask server
├── model_utils.py                  # tokenize(), get_token_trajectory(),
reduce_trajectories()
├── notebooks/
│   ├── tutorial-walkthrough.ipynb   # Task 1.2
│   ├── tokenization.ipynb           # Task 1.3
│   ├── embedding_extraction.ipynb   # Task 1.4
│   └── dimensionality_reduction.ipynb  # Task 1.5
└── README.md                        # Setup instructions
```

Week 2: API Completion (Feb 24–28)

Goal: All API endpoints working, example data generated for Polly, documentation complete.

Task 2.1: `/trace` Endpoint (~3 hrs)

Who: Steven

What to do:

- Combine the embedding extraction + PCA functions into a single endpoint:

```
POST /trace
Body: {
  "text": "The cat sat on the mat",
  "token_indices": [1, 4]
}
Response: {
  "tokens": [
    {"index": 1, "token_str": " cat"},
    {"index": 4, "token_str": " the"}
  ],
  "trajectories": {
    "1": [
      {"layer": 0, "x": -2.34, "y": 1.56},
      {"layer": 1, "x": -1.89, "y": 2.01},
      ...
    ],
    "4": [
      {"layer": 0, "x": 0.45, "y": -0.78},
      ...
    ]
  },
  "pca_explained_variance": [0.34, 0.21]
}
```

- Include the PCA explained variance ratio so the frontend knows how much information the 2D view captures
- Test with `curl` and verify the coordinates change across layers

Done when: Endpoint returns well-structured JSON with 2D coordinates for all 12 layers. Tested with 2+ different inputs.

Task 2.2: `/attention` Endpoint (~3 hrs)

Who: Mario

What to do:

- Create an endpoint that returns attention patterns for a given layer and head:

```
POST /attention
Body: {
  "text": "The cat sat on the mat",
  "layer": 5,
  "head": 3
}
Response: {
  "tokens": ["The", " cat", " sat", " on", " the", " mat"],
  "attention_matrix": [
    [0.85, 0.05, 0.03, 0.02, 0.03, 0.02],
    [0.15, 0.45, 0.20, 0.08, 0.07, 0.05],
    ...
  ]
}
```

- The attention matrix is a square matrix (n_tokens x n_tokens) where `[i][j]` is how much token `i` attends to token `j`
- Extract from cache: `cache["blocks.{layer}.attn.hook_pattern"][0, head]`
- **Bonus:** Add an optional parameter to return all heads for a layer at once

Done when: Returns correct attention matrix. Verified by checking rows sum to ~1.0 (they're softmaxed probabilities).

Task 2.3: `/predict` Endpoint — Logit Lens (~3 hrs)

Who: Steven

What to do:

- Create an endpoint that shows what the model would predict at each layer (logit lens technique):

```
POST /predict
Body: {
  "text": "The cat sat on the mat",
  "token_index": 5
}
Response: {
  "predictions_by_layer": [
    {
```

```
      "layer": 0,
      "top_tokens": [
        {"token": " the", "probability": 0.08},
        {"token": " a", "probability": 0.05},
        {"token": ",", "probability": 0.04},
        {"token": " and", "probability": 0.03},
        {"token": " of", "probability": 0.02}
      ]
    },
    {
      "layer": 1,
      "top_tokens": [...]
    },
    ...
  ]
}
```

- For each layer, project the residual stream through the unembedding matrix:

  ```
  residual = cache["resid_post", layer][0, token_index]  # (768,)
  logits = model.unembed(model.ln_final(residual.unsqueeze(0).unsqueeze(0)))  # project to vocab
  probs = torch.softmax(logits[0, 0], dim=-1)
  top5 = torch.topk(probs, 5)
  ```

- Return top 5 predicted tokens and their probabilities at each layer

Done when: Can see predictions evolve across layers. Early layers should be vague/generic, later layers should converge on the actual next token.

Resources:

- Logit lens explanation: https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens

---

Task 2.4: Generate Example Data for Frontend (~2 hrs)

Who: Mario

What to do:

- Create `examples/` folder
- Run all 4 endpoints with 3 different input texts and save the JSON responses:

1. `"The cat sat on the mat"` (simple, familiar)
2. `"Her name was Alex Hart. Tomorrow Alex"` (the induction example from the tutorial)
3. `"The Eiffel Tower is located in the city of"` (knowledge recall)

- Save as:
  - `examples/example1_tokenize.json`
  - `examples/example1_trace.json`
  - `examples/example1_attention.json`
  - `examples/example1_predict.json`
  - (repeat for examples 2 and 3)

Done when: 12 JSON files in `examples/`. Polly can use these as real mock data for the frontend without needing to run the backend.

Task 2.5: API Documentation + README (~2 hrs)

Who: Steven, Mario

What to do:

- Update `README.md` with:
  - Setup instructions (clone, create venv, install deps, run)
  - All 4 endpoints documented (URL, method, request body, response format)
  - Example `curl` commands for each endpoint
- Create `API.md` with the full JSON schemas

Done when: Someone who wasn't on the project can follow the README, start the server, and successfully call all 4 endpoints.

Week 2 File Structure (Final)

```
interpretative-interfaces-backend/
├── venv/
├── requirements.txt
├── app.py                          # Flask server with all endpoints
├── model_utils.py                  # All model functions
├── notebooks/
│   ├── tutorial-walkthrough.ipynb
│   ├── tokenization.ipynb
```

```
│   ├── embedding_extraction.ipynb
│   └── dimensionality_reduction.ipynb
├── examples/
│   ├── example1_tokenize.json
│   ├── example1_trace.json
│   ├── example1_attention.json
│   ├── example1_predict.json
│   ├── example2_*.json
│   └── example3_*.json
├── API.md                          # Endpoint documentation
└── README.md                       # Setup + usage instructions
```

Polly — Frontend/Design (Figma)

Week 1: Research + Wireframes (Feb 17–21)

Task P1.1: Study Existing Projects (~2 hrs)

What to do:

- Spend time with these 3 projects, take notes on what works and what doesn't:
  1. **Transformer Explainer** — https://poloclub.github.io/transformer-explainer/
     - Play with it for 15 min. Note: the Sankey flow diagram, the hover interactions, the temperature slider
  2. **LLM Visualization (Brendan Bycroft)** — https://bbycroft.net/llm
     - Play with it for 15 min. Note: the 3D animation, the sense of "flowing through" the architecture
  3. **Ecco documentation** — https://ecco.readthedocs.io/en/latest/
     - Look at the gallery/examples. Note: how token trajectories are visualized (2D scatter with connected paths)
- Write a short doc (1 page) with:
  - 3 things that work well across these projects
  - 3 things that are missing or could be better
  - 1 initial idea for how our project could be different

Done when: 1-page doc shared with the team.

Resources:

- Our Figma board with the examples of existing projects and brainstorming session recap.

Task P1.2: User Flow Wireframes (~3 hrs)

What to do:

- In Figma, create low-fidelity wireframes (gray boxes, no styling) for the core user flow:
  1. **Start screen:** Text input field, "Analyze" button
  2. **Token selection:** Text is displayed as individual tokens. User clicks to select 1-3 tokens (they highlight)
  3. **Trajectory view:** 2D scatter plot showing selected tokens' paths through 12 layers. Each token is a different color. Layers are connected by lines. User can hover a point to see layer number and nearby words
  4. **Attention view:** Select a layer → see the attention heatmap (which tokens attend to which). Matrix or arc diagram
  5. **Prediction view:** Select a token position → see top-5 predicted words at each layer (the logit lens). Shows how predictions evolve from vague to specific
- Don't worry about visual polish yet — focus on layout and flow

Done when: 5 wireframe screens in Figma, connected with basic click-through navigation.

Resources:

- API response schemas (coordinate with Mario/Steven for JSON structure)
- Transformer Explainer for layout inspiration

Task P1.3: Interaction Sketches (~3 hrs)

What to do:

- For the **trajectory view** (the most important screen), sketch 3 different representation ideas:
  1. **Scatter plot** — tokens as colored dots in 2D PCA space, lines connecting same token across layers (like Ecco)
  2. **Layer stack** — layers as horizontal rows, tokens move left/right as their position in semantic space shifts
  3. **One wild card** — your choice. Could be: circular/radial, 3D depth, organic/flowing, or something entirely new

- For each, annotate: how does the user select a layer? How do they compare tokens? Where do annotations go?

Done when: 3 sketches in Figma with annotations.

Week 2: Clickable Prototype (Feb 24–28)

Task P2.1: Build Clickable Prototype (~4 hrs)

What to do:

- Pick the strongest wireframe from Week 1 (or combine elements)
- Build a mid-fidelity Figma prototype with:
  - Real-ish data (use the JSON examples from Mario/Steven's `examples/` folder)
  - Click-through flow from text input → token selection → trajectory view → attention view
  - Hover states on tokens and trajectory points
  - Color system: 2-3 colors for selected tokens, neutral for background
- Use Figma's native prototyping (connections, overlays, hover states)

Done when: Someone unfamiliar with the project can click through the prototype and understand what's happening without explanation.

Task P2.2: Representation Mode Comparison (~2 hrs)

What to do:

- Take the same example data and create the trajectory view in 2 different visual styles:
  1. The "structured" mode (scatter plot or layer stack — more analytical)
  2. The "organic" mode (flowing, soft, spatial — more exploratory)
- Present them side by side with a sentence explaining the vibe of each

Done when: 2 versions of the same screen, same data, different visual language.

Task P2.3: Visual Spec Document (~2 hrs)

What to do:

- Create a Figma page called "Specs" with:
  - Color palette (token colors, background, text, interactive states)
  - Typography (font, sizes for labels, headings, data)
  - Component inventory: token chip, layer marker, trajectory line, attention cell, prediction row
  - Spacing/layout grid

Done when: A developer could look at this page and know exactly what CSS to write.

Summary: Two-Week Deliverables

Who	Week 1 Deliverable	Week 2 Deliverable
Steven	Tokenization function, PCA reduction + matplotlib plot	`/trace` endpoint, `/predict` endpoint (logit lens)
Mario	Embedding extraction function, Flask skeleton + `/tokenize`	`/attention` endpoint, example JSON data, README
Both	Tutorial walkthrough notebook, environment setup	API documentation
Polly	5 wireframe screens, 3 representation sketches, research doc	Clickable prototype, 2 visual modes, spec document

Time Budget

Steven (16 hrs total)

Task	Hours	Week
1.1 Environment setup	1	1
1.2 Tutorial walkthrough	1.5	1
1.3 Tokenization function	1.5	1
1.5 Dimensionality reduction	2.5	1
Buffer / debugging	1.5	1
2.1 `/trace` endpoint	3	2
2.3 `/predict` endpoint	3	2

Task	Hours	Week
2.5 Documentation (shared)	1	2
Buffer / debugging	1.5	2

Mario (16 hrs total)

Task	Hours	Week
1.1 Environment setup	1	1
1.2 Tutorial walkthrough	1.5	1
1.4 Embedding extraction	2	1
1.6 Flask skeleton + `/tokenize`	2	1
Buffer / debugging	1.5	1
2.2 `/attention` endpoint	3	2
2.4 Example data generation	2	2
2.5 Documentation (shared)	1	2
Buffer / debugging	2	2

Polly (16 hrs total)

Task	Hours	Week
P1.1 Research existing projects	2	1
P1.2 User flow wireframes	3	1
P1.3 Representation sketches	3	1
P2.1 Clickable prototype	4	2
P2.2 Representation comparison	2	2
P2.3 Visual spec document	2	2

Blocker Protocol

⚠ **If you're stuck, message Gabrielle.**

Seek help when debugging. Common blockers:

- **TransformerLens won't install:** Try `pip install git+https://github.com/TransformerLensOrg/TransformerLens.git`
- **Model download fails:** Check internet connection, try a different network
- **Shape mismatch errors:** Print `.shape` of every tensor. GPT-2 small dimensions: vocab=50257, d_model=768, n_layers=12, n_heads=12
- **Flask CORS errors:** Make sure `flask-cors` is installed and `CORS(app)` is in `app.py`

Created: 2026-02-15
Sprint: Feb 17 – Feb 28, 2026