Structured Perspective Generator (Vertex AI Endpoint Only)

Generates a set of structured "perspectives" (default 70) over evenly distributed bias/significance coordinates using **your deployed Vertex AI endpoint**. Public Gemini API key mode has been removed for this project.

Features

- Vertex endpoint only (enforced)
- Input specification via `input.json`
- Even distribution of `bias_x` and `significance_y` in [0,1]
- Color assignment across N colors (default 7)
- Model instructed to output strict JSON only
- Robust JSON extraction (falls back to scaffold if parse fails)
- Raw streamed text saved to `raw_model_output.txt`

Quick Start (Vertex AI)

```
python -m venv .venv
.venv\Scripts\Activate.ps1
pip install -r requirements.txt
Copy-Item .env.example .env
# Edit .env and set VERTEX_ENDPOINT only (ensure ADC auth configured)
gcloud auth application-default login
python api_request.py --input input.json --output output.json --count 70 --
colors 7 --endpoint projects/<project>/locations/<region>/endpoints/<endpoint-
id>
```

Environment Variables

The script loads `.env` if present.

Variable	Purpose
`VERTEX_ENDPOINT`	Required endpoint path (can be overridden with --endpoint)
`GOOGLE_APPLICATION_CREDENTIALS`	(Optional) Path to service account key if not using gcloud ADC login , Format: projects//locations//endpoints/
`DEFAULT_TEMPERATURE`	Overrides default temperature if provided
`PROMPT_SUFFIX`	Extra instruction appended to prompt (optional)

Authenticate (one of):

```
gcloud auth application-default login
# OR
$env:GOOGLE_APPLICATION_CREDENTIALS = "C:\path\to\service-account.json"
```

Input File (`input.json`)

Example already included:

```
{
  "topic": "Sustainable Urban Mobility",
  "context": "A mid-sized European city wants to reduce congestion and
emissions while improving accessibility for residents and visitors.",
  "objectives": ["Lower CO2 emissions by 40% in 5 years", "Increase public
transit ridership by 25%"],
  "constraints": ["Limited capital budget in first 2 years"],
  "stakeholders": ["Residents", "Local businesses", "Tourists"]
}
```

Output

`output.json` will contain an array of perspective objects:

```
[
  {
    "index": 0,
    "color": "color_1",
    "bias_x": 0.0,
    "significance_y": 0.0,
    "title": "...",
    "perspective": "...",
    "impact_score": 0.73,
    "significance_explanation": "...",
    "risks": ["..."],
    "action_hint": "..."
  }
]
```

If parsing fails, the scaffold (numeric fields only) is written instead and a warning is printed.

CLI Arguments

Flag	Default	Description
`--input`	`input.json`	Input file path
`--output`	`output.json`	Output file path

Flag	Default	Description
`--endpoint`	from `VERTEX_ENDPOINT` env	Vertex endpoint path (required if env not set)
`--model`	(deprecated)	Backwards compat; treated as endpoint if provided
`--count`	70	Number of perspectives
`--colors`	7	Number of colors
`--temperature`	0.6	Sampling temperature

Troubleshooting

- Endpoint pattern error: Ensure it matches `projects/<project>/locations/<region>/endpoints/<id>`.
- Credential errors: Run `gcloud auth application-default login` or set `GOOGLE_APPLICATION_CREDENTIALS`.
- Empty / malformed JSON: Inspect `raw_model_output.txt`.

Possible Enhancements

- Add retry/backoff logic
- Add schema validation (e.g., with `pydantic`)
- Stream partial JSON to file progressively
- Wire `PROMPT_SUFFIX` into prompt construction

PRs or suggestions welcome.

Frontend

React app located in `frontend/`.

Aceternity (shadcn/ui) Integration

The project uses Tailwind CSS and has been manually prepared for shadcn/ui (Aceternity UI) components.

Manual setup steps performed:

1. Installed Tailwind (v3) with PostCSS + Autoprefixer.
2. Added design tokens and utility layers to `frontend/src/index.css`.
3. Extended `frontend/tailwind.config.js` with CSS variable-driven color system and animation keyframes.
4. Installed supporting libraries: `clsx`, `tailwind-merge`, `class-variance-authority`, `lucide-react`, `@radix-ui/react-slot`, `tailwindcss-animate`.

Add a new component (manual method):

1. Visit https://ui.shadcn.com
2. Select the component you want.

3. Copy the component's source (including any Radix primitives) into `frontend/src/components/ui/<ComponentName>.jsx`.
4. Ensure any referenced utilities (e.g. `cn`) are imported from `frontend/src/lib/utils.js`.

Example utility import:

```
import { cn } from '../lib/utils';
```

Dark mode: add `class="dark"` on `html` or a wrapper to switch theme.

Animation helpers come from `tailwindcss-animate` (already configured).

If/when the official CLI detects CRA properly you can attempt:

```
npx shadcn@latest add button
```

Otherwise continue using manual copy/paste.

Background Ripple Effect

Implemented via `BackgroundRippleEffect` (`frontend/src/components/ui/background-ripple-effect.js`). Added globally in `App.js` behind content with layering (`z-0` vs content `z-10`).

Customization props:

```
<BackgroundRippleEffect rows={8} cols={27} cellSize={56} auto idle
idleInterval={4000} />
```

Props:

- `auto` (default true): fire an initial centered ripple on mount.
- `idle` (default true): periodically trigger random ripples.
- `idleInterval` ms between idle ripples (default 4000).
  Performance: Larger grids raise layout cost; reduce `cols` or `rows` or increase `cellSize` for cheaper paint.

Disable globally: remove the component from `App.js` or wrap in a feature flag.

Animation parameters (delay & duration) are derived from ripple distance; keyframes defined in `index.css` under `@layer utilities`.

Styling Guide (Tokens + Utilities)

The frontend now follows a token + utility approach inspired by shadcn/ui (Aceternity) patterns:

1. Design Tokens: Defined as CSS variables in `index.css` (`--background`, `--foreground`, `--card`, etc.). Always prefer `hsl(var(--token))` over hard-coding hex values.
2. Tailwind Utilities: Prefer composing utilities (`p-5 rounded-lg border border-border bg-card/30`) instead of inline `style={{...}}`. Inline styles are reserved only for dynamic values not easily represented by classes (e.g. computed gradient strings).
3. Gradients: Keep semantic gradients in JS helpers (e.g. `colorGradient`) but let surrounding surfaces use tokenized backgrounds.
4. Glass / Blur Surfaces: Use `bg-card/30 backdrop-blur-sm` (+ optional border) for translucent panels.
5. Shadows: Favor subtle layered depth: `shadow-sm`, optionally custom box-shadows only when meaningfully distinct.
6. State Indicators: Use utility classes + existing color scale (`bg-primary`, `text-muted-foreground`, pulse indicators) instead of ad-hoc colored text.
7. Accessibility: Maintain focus visibility via Tailwind ring utilities (`focus-visible:ring-ring`). Avoid removing outlines.
8. Component Cohesion: Shared primitives (e.g. Button) live in `components/ui/`. New primitives should accept `className` and use the `cn` helper.

Refactored:

- Removed most inline styling from `Component3` subcomponents (`KnnProcessingDisplay`, `PerspectiveDisplay`, `ScatterPlotChart`, `SequentialDisplay`).
- Tokenized `ExpandablePerspectiveCards.css` (foreground/background/border/ring) replacing raw `#fff` / `#111`.
- Added blur + subtle translucency to dialogs/cards to align with modern Aceternity aesthetic.

When adding new UI components:

1. Start with semantic structure in JSX.
2. Apply Tailwind classes using tokens (e.g. `bg-card text-foreground`).
3. Extract repeated style patterns into small components or variant objects (similar to `button.js`).
4. Only introduce new CSS if: complex animation, keyframes, or stateful layout effect not covered by utilities.

If you need a new color surface, extend tokens in `:root` + `tailwind.config.js` rather than scattering new hex codes.

TODO / Potential Enhancements

Area	Idea
Cards	Add keyboard focus trap for dialog (currently basic ESC + outside click)
Theme	Add light theme toggle component that toggles `class="dark"` on root
Motion	Extract motion transitions into a shared config for consistency
Docs	Generate a component catalog page with examples