```
\documentclass[12pt,a4paper]{article}
\usepackage[utf8]{inputenc}
\usepackage[english]{babel}
\usepackage[margin=2.5cm]{geometry}
\usepackage{inconsolata} % Pleasant monospaced font
\usepackage{hyperref}
\hypersetup{colorlinks=true, linkcolor=MidnightBlue, urlcolor=MidnightBlue}
\usepackage{fancyvrb} % for displaying code

\title{PROJECT ASSIMOV: A Manifesto for Educators in the Age of Transformers}
\author{Dídac Valenciano Gener}
\date{May 2025}

\begin{document}
\section*{2025 is — and will be — the year of ChatGPT.}
```

In Iberian and Mediterranean Europe —still shaped by a latent mecha subconscious— *Mazinger Z* left a cultural and emotional imprint that marked an entire generation. For those aged 45 to 60, it's all still there: the battle cry "*¡Puños fuera!*", the survival of *Blandiblú*, the *Famobil Clicks*, and afternoons filled with violent cartoons — without turning into psychopaths.

In the United States, *Hasbro* led a generational shift with the rise of the *Transformers*. The saga hit its cultural peak in 2007, when, in the opening scenes, the *Decepticons* attacked a U.S. military base in the Qatari desert.

Megan Fox and *Shia LaBeouf* stole the spotlight — but something else stayed with us.

A small *Decepticon* — a *Transformer*, but one of the bad ones — hides in plain sight among the protagonists' things. It absorbs metal, mutates, stretches, elongates, adapts. Until it becomes what it needs to be.

Ten years later, as the result of a long research journey — by both academic and private entities — and in parallel with the entertainment world, *Google* publicly introduced, in 2017, an algorithmic technology.

One that replicates itself. Stretches. Expands its own structure. Modifies and elongates — until, purely by probability, it reaches its goal.

What a surprise — and what a coincidence — that this technology, previously unknown, was named: *transformers*.

And the beauty of it, my friends, is this: it simulates so well.

If we define *artificial intelligence* as a functional simulation of intelligence... well, there might be other implementations.

But this one — definitely — is one.

What follows is another implementation.

One that, like the algorithms from *OpenAI*, stretches, adapts, and reshapes itself — all to generate, in a purely probabilistic way, the best possible answer to the task ahead.

In this case? Creating content. Didactic material. Articles. Maybe even a book.

So that we, *teachers*, have our own tools — ready for the battle that lies ahead.

We got all the weapons we need: Now fight! *(Sucker Punch — 2011)*

AI-enhanced LaTeX generation pipeline

github.com/nassaba/project-assimov - AI-enhanced content generation pipeline

```python
# Phase 0: Interpret prompt and assign roles with
    grammarnaut + llm_router

# Phase 1: Generate structural skeleton with yaml_generator
    using Claude or o3 or ...

# Phase 2: Expand each section with role-assigned LLMs (GPT
    -4o, MythoMax, Lit-6B...) using cached YAMLs

# Phase 3: Review narrative coherence, argument and tone
    with o3 or Claude using section context

# Phase 4: Compile formatted output into .tex with writer or
     translator

# Phase 5 (optional): Final stylistic polish by purist LLM
    or human reviewer

# CONFIGURATION
# Load API key (replace with your secure method)

import openai
from pathlib import Path
import time


# AI-enhanced content generation pipeline (Assimov)

def phase_0_prompt_intake():
    """Receive user prompt and determine functional roles +
       LLMs."""
    prompt = user_input()
    roles = assign_roles(prompt)  # via gramaneute.py ->
       llm_router.py
    llms = select_models(roles, config="config.yaml")
    return plan(roles, llms)

def phase_1_generate_structure(plan):
    """Create YAML skeleton per section with titles, themes,
        targets."""
    yamls = []
    for section in plan.sections:
    yamls.append(generate_yaml(section))  # via
       yaml_generator.py or real LLM
    return yamls

```

```python
def phase_2_expand_sections(yamls):
    """Expand each section via assigned LLM, using cached context."""
    content = []
    for yaml in yamls:
        llm = yaml.assigned_model
        section_text = expand_from_yaml(yaml, llm=llm, use_cache=True)
        content.append(section_text)
    return content

def phase_3_refine_coherence(content):
    """Polish narrative flow, argument structure, and tonal coherence."""
    polished = []
    for section in content:
        coherent = enforce_coherence(section)  # coherence supervisor
        refined = polish_argument(coherent)
        toned = adjust_tone(refined)
        polished.append(toned)
    return polished

def phase_4_compile_tex(polished):
    """Format polished content into a .tex document with structure."""
    doc = initialize_tex()
    for section in polished:
        doc.append(format_section(section))  # via writer.py or translator.py
    return doc

def phase_5_final_review(tex_document):
    """Optional: stylistic and poetic pass by a purist LLM or human."""
    reviewed = manual_review(tex_document)
    return reviewed

# Main pipeline execution

if __name__ == "__main__":
    plan = phase_0_prompt_intake()
    yamls = phase_1_generate_structure(plan)
    raw_content = phase_2_expand_sections(yamls)
    refined = phase_3_refine_coherence(raw_content)
    tex = phase_4_compile_tex(refined)
    final_output = phase_5_final_review(tex)
    save(final_output, "output/document.tex")
```