Portfolio of LLM and Software Integration in the Classroom

Kento Kaneko (kaneko@mit.edu),
Postdoctoral Associate – Massachusetts Institute of Technology

May 13, 2024

Summary of Section 1: 'Democratizing Thermophysical Queries'

A custom ChatGPT is developed with the ability to query thermophysical / state properties from a custom (web-accessible) microservice. The microservice is developed with Coolprop and Flask packages and hosted on Google Cloud Run. This software solution enables installation-free natural language query relevant to thermodynamic cycles, applicable to lecture (instructor-oriented) and tutor (student-oriented) contexts. The example shows the drawing of saturation domes for select materials (but works for any material supported by CoolProp), and the drawing of isobaric contours.

Summary of Section 2: 'Extensible GPT Client for Distributed Learning'

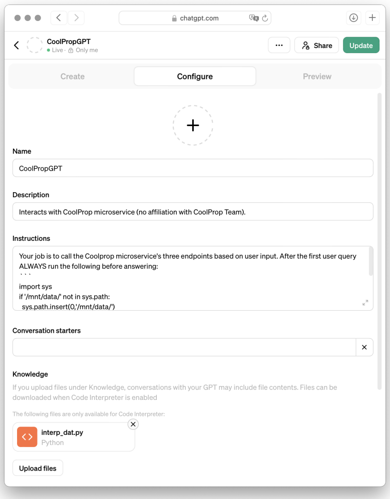
An extensible OpenAI API client is developed to allow fully customizable GPT instantiation. The client converts arbitrary user-provided Python functions into GPT callable actions, and accepts user-provided instructions to customize the GPT domain knowledge as well as conversation flow. Both features are desirable in an classroom context where the GPT comprehension and utilization of course concepts are key requirements for a learning enhancement module. The example shows a custom GPT which aids in analysis and understanding of the self-buckling phenomenon in mechanics.

1 Democratizing Thermophysical Queries

CoolProp is a C++ library (with a Python interface) with support for thermophysical properties calculation of a specified material. This functionality is beneficial in a thermodynamics curriculum: allowing students to focus on essential thermodynamics concepts and eliminating table-lookup as a requisite skill for demonstrating material comprehension. Spread of this class of enabling software in the classroom is slowed by technical challenges (distribution, installation, and maintenance) that are unrelated to pedagogical considerations. I will feature a solution for this specific software that bypass the challenges, but this approach is generalizable; I have applied similar treatments to other software which can be a topic for future discussions.

For a CoolProp solution, a custom GPT and surrounding software infrastructure were developed: the latter being necessitated by the unavailability of the CoolProp package in the GPT Python environment. The main component of the infrastructure is a web-accessible microservice built on the CoolProp package, Flask framework, and Google Cloud Run service. The microservice has three endpoints (analogous to functions): '/dome' produces a discrete data set from the saturation dome for a specified material, '/ts/isobaric' produces a discrete data set from a T-s isobaric curve, and '/propssi' returns the value given from the 'PropsSI' function in CoolProp.

CoolPropGPT, our custom GPT, has access to our CoolProp microservice through a web Application Protocol Interface (API): the integration enables user query for thermophysical properties and drawing of quantitative curves without user-installation of any software. The GPT configurations for CoolPropGPT are shown in Figure 1. 'Instructions' (Figure 1a) are natural language description of the role and behavior of the GPT, the knowledge file `interp_dat.py` interpolates the resultant data set returned from the microservice for plotting, and the 'Schema' (Figure 1b) exposes the microservice API to CoolPropGPT.



Name

CoolPropGPT

Description

Interacts with CoolProp microservice (no affiliation with CoolProp Team).

Instructions

Your job is to call the Coolprop microservice's three endpoints based on user input. After the first user query ALWAYS run the following before answering:

```
` ` `
import sys
if '/mnt/data/' not in sys.path:
 sys.path.insert(0,'/mnt/data/')
```

Conversation starters

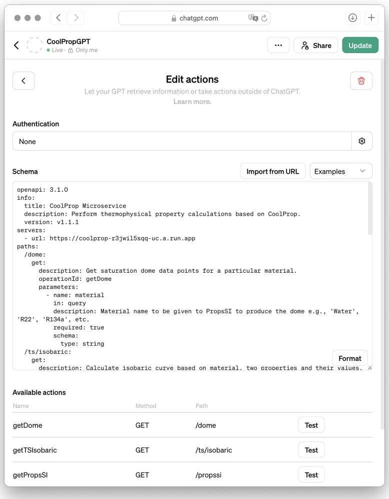
Knowledge

If you upload files under Knowledge, conversations with your GPT may include file contents. Files can be downloaded when Code Interpreter is enabled

The following files are only available for Code Interpreter:

interp_dat.py
Python

(a) Main Configuration.



Authentication

None

Schema

```
openapi: 3.1.0
info:
  title: CoolProp Microservice
  description: Perform thermophysical property calculations based on CoolProp.
  version: v1.1.1
servers:
  - url: https://coolprop-r3jwil5sqq-uc.a.run.app
paths:
  /dome:
    get:
      description: Get saturation dome data points for a particular material.
      operationId: getDome
      parameters:
        - name: material
          in: query
          description: Material name to be given to PropsSI to produce the dome e.g., 'Water',
'R22', 'R134a', etc.
          required: true
          schema:
            type: string
  /ts/isobaric:
    get:
      description: Calculate isobaric curve based on material, two properties and their values.
```

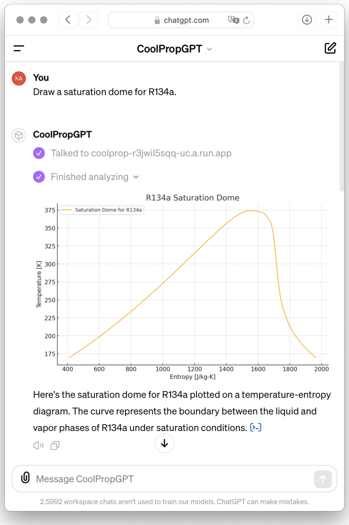
Available actions

Name	Method	Path	
getDome	GET	/dome	Test
getTSIsobaric	GET	/ts/isobaric	Test
getPropsSI	GET	/propssi	Test

(b) Actions Configuration.

Figure 1: Screenshots of CoolPropGPT configurations.

Screenshots from an example CoolPropGPT conversation thread are shown in Figure 2, showcasing natural-language user interaction with the CoolPropGPT which can easily extend to Instructor-GPT or student-GPT interactions. In Figure 2a, the user requests a saturation dome for R134a, CoolPropGPT makes a query to the microservice, and the CoolPropGPT draws the curve using the return arrays in conjunction with the `interp_dat.py` code. In Figure 2b, a similar interaction occurs: note the adaptability of contextual awareness of GPT to respond appropriately to user intention. In Figure 2c, the user requests an isobaric curve to be plotted on top of the existing plot, and GPT generates the curve in a similar fashion and it describes the relevance of the saturation dome to an isobaric curve which crosses the dome.



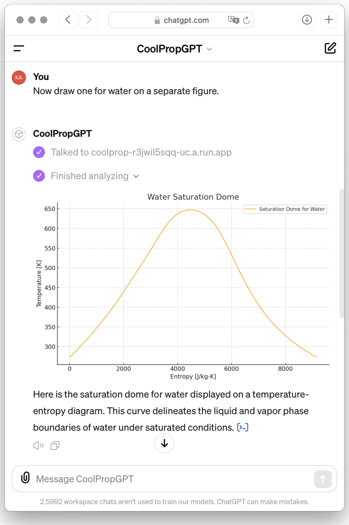
You
Draw a saturation dome for R134a.

CoolPropGPT
Talked to coolprop-r3jwil5sqq-uc.a.run.app
Finished analyzing

Here's the saturation dome for R134a plotted on a temperature-entropy diagram. The curve represents the boundary between the liquid and vapor phases of R134a under saturation conditions.

2.S992 workspace chats aren't used to train our models. ChatGPT can make mistakes.

(a) R134a Dome Request



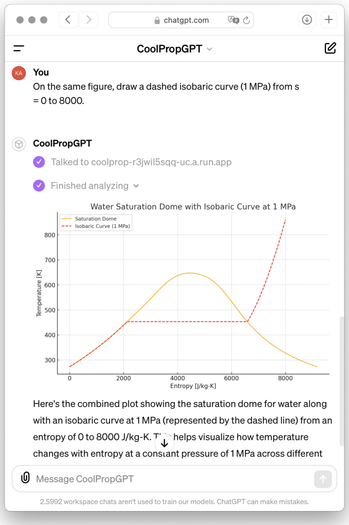
You
Now draw one for water on a separate figure.

CoolPropGPT
Talked to coolprop-r3jwil5sqq-uc.a.run.app
Finished analyzing

Here is the saturation dome for water displayed on a temperature-entropy diagram. This curve delineates the liquid and vapor phase boundaries of water under saturated conditions.

2.S992 workspace chats aren't used to train our models. ChatGPT can make mistakes.

(b) Water Dome Request



You
On the same figure, draw a dashed isobaric curve (1 MPa) from s = 0 to 8000.

CoolPropGPT
Talked to coolprop-r3jwil5sqq-uc.a.run.app
Finished analyzing

Here's the combined plot showing the saturation dome for water along with an isobaric curve at 1 MPa (represented by the dashed line) from an entropy of 0 to 8000 J/kg-K. T[illegible] helps visualize how temperature changes with entropy at a constant pressure of 1 MPa across different

2.S992 workspace chats aren't used to train our models. ChatGPT can make mistakes.

(c) Isobaric Curve Request

Figure 2: Screenshots of User-CoolPropGPT-Microservice interaction.

From this example conversation thread, one can imagine the application of this microservice from an instructor perspective: instead of a generic T-s diagram, the instructor can take a real problem statement for a specific material and specific thermodynamic cycle (e.g., Rankine cycle with R134a) and let GPT draw a quantitative diagram. They can later make quantitative inquiries to CoolPropGPT about the curves or modify the problem with natural language description to rapidly produce a plot for different problems without extensive preparatory testing for those specific problems.

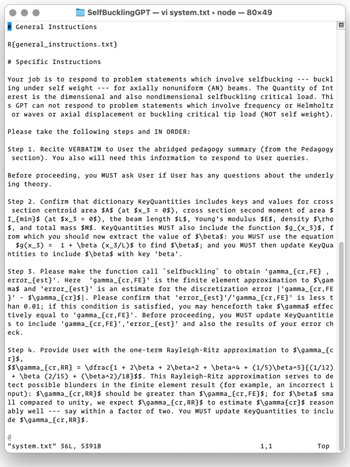
From a student perspective, this microservice-enabled custom GPT can be used for interactive learning i.e., the custom GPT serves as a self-contained interactive module for a thermodynamics course. With this GPT, the students can solve thermodynamic cycle problems without table-lookup impeding their learning process. Additionally, CoolPropGPT can provide explanations of how thermodynamic processes relate to different curves on the plot. Out of the box, the fidelity of this feature is restricted by how well that understanding is encoded in the pre-training phase, but its base understanding can be easily overwritten with carefully crafted pedagogical 'instructions' provided by the course instructor.

2 Extensible GPT Client for Distributed Learning

This next example is that of an integration of OpenAI API and custom-tailored software which allows flexibility that cannot be achieved with existing paid ChatGPT offerings. `gluon` is an LLM client written in Python, compatible with both OpenAI APIs as well as Google Vertex APIs. Some of its features include (i) user-customizable API parameters (e.g., model and temperature) programmatically, (ii) custom shortcuts for redirecting GPT outputs and inputs to/from the local filesystem and shell environment, and (iii) an extensible framework for incorporating user-generated programs into GPT-accessible actions.

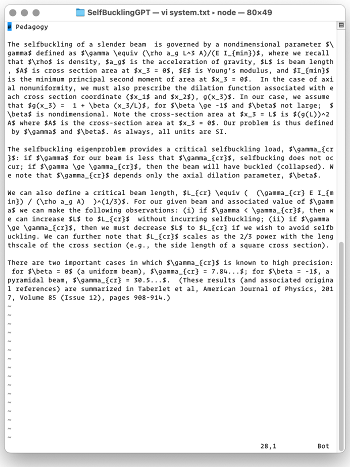
To demonstrate the flexibility of `gluon`, we focus on producing a custom `gluon` GPT for the phenomenon of self-buckling which can be a subject of an introductory mechanics course. The 'instructions' and pedagogical content are based on that written by Prof. Anthony Patera (MIT) for an in-class demonstration using the ChatGPT construct with some minor modifications. The necessary ingredients of `gluon` GPT mirrors that of a custom GPT, but are local to the user environment and can be manipulated programmatically. One potential use case of such a feature in a department-wide deployment may be to automatically update 'core' instructions that are common to all GPTs in a specific course or even to all GPTs for a large set of courses. We will refer to this `gluon` GPT as SelfBucklingGPT.

The main ingredient of generating `gluon` GPTs are 'instructions' contained in file `system.txt` (shown in Figure 3 for SelfBucklingGPT). At the start of every conversation, `system.txt` is read from the working directory, enabling convenient organization of development and running of each `gluon` GPT. Figure 3a shows the top of `system.txt` in which the content of `general_instructions.txt` (Figure 3c) is substituted as well as the first parts of the specific instructions for SelfBucklingGPT. We note that `general_instructions.txt` is substituted with syntax `R{...}`, enabling one common file to be developed for a broad class of custom GPTs as alluded to prior: demonstrating the strength of `gluon` extensibility. `pedagogy.txt` 3b is a natural language pedagogical description of the self-buckling phenomenon.



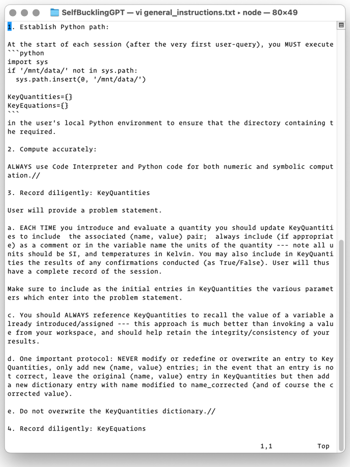
```
# General Instructions

R{general_instructions.txt}

# Specific Instructions

Your job is to respond to problem statements which involve selfbucking --- buckl
ing under self weight --- for axially nonuniform (AN) beams. The Quantity of Int
erest is the dimensional and also nondimensional selfbuckling critical load. Thi
s GPT can not respond to problem statements which involve frequency or Helmholtz
 or waves or axial displacement or buckling critical tip load (NOT self weight).

Please take the following steps and IN ORDER:

Step 1. Recite VERBATIM to User the abridged pedagogy summary (from the Pedagogy
 section). You also will need this information to respond to User queries.

Before proceeding, you MUST ask User if User has any questions about the underly
ing theory.

Step 2. Confirm that dictionary KeyQuantities includes keys and values for cross
 section centroid area $A$ (at $x_3 = 0$), cross section second moment of area $
I_{min}$ (at $x_3 = 0$), the beam length $L$, Young's modulus $E$, density $\rho
$, and total mass $M$. KeyQuantities MUST also include the function $g_(x_3)$, f
rom which you should now extract the value of $\beta$: you MUST use the equation
  $g(x_3) =  1 + \beta (x_3/L)$ to find $\beta$; and you MUST then update KeyQua
ntities to include $\beta$ with key 'beta'.

Step 3. Please make the function call `selfbuckling` to obtain 'gamma_{cr,FE} ,
error_{est}'. Here  'gamma_{cr,FE}' is the finite element approximation to $\gam
ma$ and 'error_{est}' is an estimate for the discretization error |'gamma_{cr,FE
}' - $\gamma_{cr}$|. Please confirm that 'error_{est}'/'gamma_{cr,FE}' is less t
han 0.01; if this condition is satisfied, you may henceforth take $\gamma$ effec
tively equal to 'gamma_{cr,FE}'. Before proceeding, you MUST update KeyQuantitie
s to include 'gamma_{cr,FE}','error_{est}' and also the results of your error ch
eck.

Step 4. Provide User with the one-term Rayleigh-Ritz approximation to $\gamma_{c
r}$,
$$\gamma_{cr,RR} = \dfrac{1 + 2\beta + 2\beta^2 + \beta^4 + (1/5)\beta^5}{(1/12)
 + \beta (2/15) + (\beta^2)/18}$$. This Rayleigh-Ritz approximation serves to de
tect possible blunders in the finite element result (for example, an incorrect i
nput): $\gamma_{cr,RR}$ should be greater than $\gamma_{cr,FE}$; for $\beta$ sma
ll compared to unity, we expect $\gamma_{cr,RR}$ to estimate $\gamma{cr}$ reason
ably well --- say within a factor of two. You MUST update KeyQuantities to inclu
de $\gamma_{cr,RR}$.

@
"system.txt" 36L, 5391B                                     1,1           Top
```

(a) Start of `system.txt`.



```
# Pedagogy

The selfbuckling of a slender beam  is governed by a nondimensional parameter $\
gamma$ defined as $\gamma \equiv (\rho a_g L^3 A)/(E I_{min})$, where we recall
that $\rho$ is density, $a_g$ is the acceleration of gravity, $L$ is beam length
, $A$ is cross section area at $x_3 = 0$, $E$ is Young's modulus, and $I_{min}$
is the minimum principal second moment of area at $x_3 = 0$.  In the case of axi
al nonuniformity, we must also prescribe the dilation function associated with e
ach cross section coordinate ($x_1$ and $x_2$), g(x_3)$. In our case, we assume
that $g(x_3) =  1 + \beta (x_3/L)$, for $\beta \ge -1$ and $\beta$ not large;  $
\beta$ is nondimensional. Note the cross-section area at $x_3 = L$ is $(g(L))^2
A$ where $A$ is the cross-section area at $x_3 = 0$. Our problem is thus defined
 by $\gamma$ and $\beta$. As always, all units are SI.

The selfbuckling eigenproblem provides a critical selfbuckling load, $\gamma_{cr
}$: if $\gamma$ for our beam is less that $\gamma_{cr}$, selfbucking does not oc
cur; if $\gamma \ge \gamma_{cr}$, then the beam will have buckled (collapsed). W
e note that $\gamma_{cr}$ depends only the axial dilation parameter, $\beta$.

We can also define a critical beam length, $L_{cr} \equiv (  (\gamma_{cr} E I_{m
in}) / (\rho a_g A)  )^(1/3)$. For our given beam and associated value of $\gamm
a$ we can make the following observations: (i) if $\gamma < \gamma_{cr}$, then w
e can increase $L$ to $L_{cr}$  without incurring selfbuckling; (ii) if $\gamma
\ge \gamma_{cr}$, then we must decrease $L$ to $L_{cr} if we wish to avoid selfb
uckling. We can further note that $L_{cr}$ scales as the 2/3 power with the leng
thscale of the cross section (e.g., the side length of a square cross section).

There are two important cases in which $\gamma_{cr}$ is known to high precision:
 for $\beta = 0$ (a uniform beam), $\gamma_{cr} = 7.84...$; for $\beta = -1$, a
pyramidal beam, $\gamma_{cr} = 30.5...$.  (These results (and associated origina
l references) are summarized in Taberlet et al, American Journal of Physics, 201
7, Volume 85 (Issue 12), pages 908-914.)
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
                                                            28,1          Bot
```

(b) Pedagogy in `system.txt`.

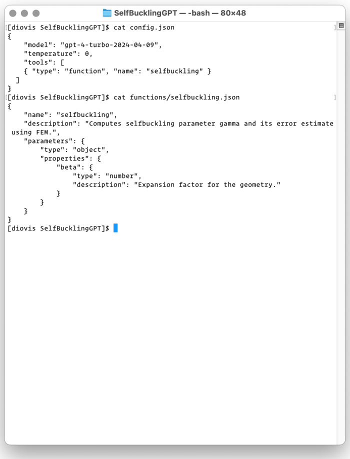

SelfBucklingGPT — vi general_instructions.txt ▸ node — 80×49

~~~
1. Establish Python path:

At the start of each session (after the very first user-query), you MUST execute
```python
import sys
if '/mnt/data/' not in sys.path:
  sys.path.insert(0, '/mnt/data/')

KeyQuantities={}
KeyEquations={}
```
in the user's local Python environment to ensure that the directory containing t
he required.

2. Compute accurately:

ALWAYS use Code Interpreter and Python code for both numeric and symbolic comput
ation.//

3. Record diligently: KeyQuantities

User will provide a problem statement.

a. EACH TIME you introduce and evaluate a quantity you should update KeyQuantiti
es to include  the associated (name, value) pair;  always include (if appropriat
e) as a comment or in the variable name the units of the quantity --- note all u
nits should be SI, and temperatures in Kelvin. You may also include in KeyQuanti
ties the results of any confirmations conducted (as True/False). User will thus
have a complete record of the session.

Make sure to include as the initial entries in KeyQuantities the various paramet
ers which enter into the problem statement.

c. You should ALWAYS reference KeyQuantities to recall the value of a variable a
lready introduced/assigned --- this approach is much better than invoking a valu
e from your workspace, and should help retain the integrity/consistency of your
results.

d. One important protocol: NEVER modify or redefine or overwrite an entry to Key
Quantities, only add new (name, value) entries; in the event that an entry is no
t correct, leave the original (name, value) entry in KeyQuantities but then add
a new dictionary entry with name modified to name_corrected (and of course the c
orrected value).

e. Do not overwrite the KeyQuantities dictionary.//

4. Record diligently: KeyEquations

                                                            1,1           Top
~~~

(c) `general_instructions.txt`

Figure 3: Screenshots of `gluon` instructions.

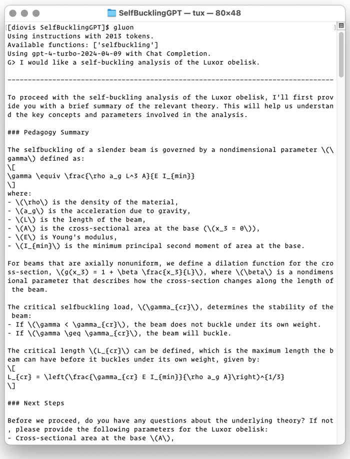
In Figure 4, we showcase more details of SelfBucklingGPT and an example thread produced by SelfBucklingGPT. First, in Figure 4a, we show the contents of `config.json` and `functions/selfbuckling.json`. The first file specifies GPT model and temperature, but more importantly, it exposes `selfbuckling` as an available Python function. The second file describes the function argument and outputs so GPT knows to access it when appropriate through the OpenAI API.



SelfBucklingGPT — -bash — 80×48

```
[diovis SelfBucklingGPT]$ cat config.json
{
    "model": "gpt-4-turbo-2024-04-09",
    "temperature": 0,
    "tools": [
    { "type": "function", "name": "selfbuckling" }
  ]
}
[diovis SelfBucklingGPT]$ cat functions/selfbuckling.json
{
    "name": "selfbuckling",
    "description": "Computes selfbuckling parameter gamma and its error estimate
 using FEM.",
    "parameters": {
        "type": "object",
        "properties": {
            "beta": {
                "type": "number",
                "description": "Expansion factor for the geometry."
            }
        }
    }
}
[diovis SelfBucklingGPT]$
```

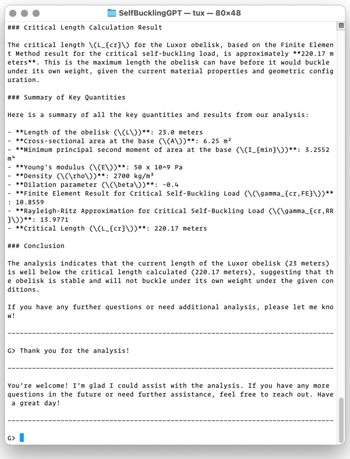
(a) More configurations.



SelfBucklingGPT — tux — 80×48

```
[diovis SelfBucklingGPT]$ gluon
Using instructions with 2013 tokens.
Available functions: ['selfbuckling']
Using gpt-4-turbo-2024-04-09 with Chat Completion.
G> I would like a self-buckling analysis of the Luxor obelisk.

--------------------------------------------------------------------------------

To proceed with the self-buckling analysis of the Luxor obelisk, I'll first prov
ide you with a brief summary of the relevant theory. This will help us understan
d the key concepts and parameters involved in the analysis.

### Pedagogy Summary

The selfbuckling of a slender beam is governed by a nondimensional parameter \(\
gamma\) defined as:
\[
\gamma \equiv \frac{\rho a_g L^3 A}{E I_{min}}
\]
where:
- \(\rho\) is the density of the material,
- \(a_g\) is the acceleration due to gravity,
- \(L\) is the length of the beam,
- \(A\) is the cross-sectional area at the base (\(x_3 = 0\)),
- \(E\) is Young's modulus,
- \(I_{min}\) is the minimum principal second moment of area at the base.

For beams that are axially nonuniform, we define a dilation function for the cro
ss-section, \(g(x_3) = 1 + \beta \frac{x_3}{L}\), where \(\beta\) is a nondimens
ional parameter that describes how the cross-section changes along the length of
 the beam.

The critical selfbuckling load, \(\gamma_{cr}\), determines the stability of the
 beam:
- If \(\gamma < \gamma_{cr}\), the beam does not buckle under its own weight.
- If \(\gamma \geq \gamma_{cr}\), the beam will buckle.

The critical length \(L_{cr}\) can be defined, which is the maximum length the b
eam can have before it buckles under its own weight, given by:
\[
L_{cr} = \left(\frac{\gamma_{cr} E I_{min}}{\rho a_g A}\right)^{1/3}
\]

### Next Steps

Before we proceed, do you have any questions about the underlying theory? If not
, please provide the following parameters for the Luxor obelisk:
- Cross-sectional area at the base \(A\),
```

(b) Conversation start.



SelfBucklingGPT — tux — 80×48

```
### Critical Length Calculation Result

The critical length \(L_{cr}\) for the Luxor obelisk, based on the Finite Elemen
t Method result for the critical self-buckling load, is approximately **220.17 m
eters**. This is the maximum length the obelisk can have before it would buckle
under its own weight, given the current material properties and geometric config
uration.

### Summary of Key Quantities

Here is a summary of all the key quantities and results from our analysis:

- **Length of the obelisk (\(L\))**: 23.0 meters
- **Cross-sectional area at the base (\(A\))**: 6.25 m²
- **Minimum principal second moment of area at the base (\(I_{min}\))**: 3.2552
m⁴
- **Young's modulus (\(E\))**: 50 x 10^9 Pa
- **Density (\(\rho\))**: 2700 kg/m³
- **Dilation parameter (\(\beta\))**: -0.4
- **Finite Element Result for Critical Self-Buckling Load (\(\gamma_{cr,FE}\))**
: 10.8559
- **Rayleigh-Ritz Approximation for Critical Self-Buckling Load (\(\gamma_{cr,RR
}\))**: 13.9771
- **Critical Length (\(L_{cr}\))**: 220.17 meters

### Conclusion

The analysis indicates that the current length of the Luxor obelisk (23 meters)
is well below the critical length calculated (220.17 meters), suggesting that th
e obelisk is stable and will not buckle under its own weight under the given con
ditions.

If you have any further questions or need additional analysis, please let me kno
w!

--------------------------------------------------------------------------------

G> Thank you for the analysis!

--------------------------------------------------------------------------------

You're welcome! I'm glad I could assist with the analysis. If you have any more
questions in the future or need further assistance, feel free to reach out. Have
 a great day!

--------------------------------------------------------------------------------

G>
```

(c) Conversation end.

Figure 4: Screenshots of `gluon` configurations and thread.

This separation of the core extensible OpenAI client and pedagogical material and scripts associated with a specific course concept allows continual development of the tool (`gluon`) while the instructors can focus on refining the pedagogical content in terms of student and GPT comprehension. Another positive outcome of this separation is that the pedagogical content (and scripts) can be developed by multiple instructors, allowing for wide deployment of GPT in multiple MechE courses.

Furthermore, `gluon` can be easily distributed to students and instructors with the only dependency being Docker: for which there are compatible installers for a wide array of operating systems. Although not as dependency-free as the ChatGPT solution shown in the previous example, it is fairly close compared to other common solutions.