Auto-BG: The Board Game Concept Generator

A Gentle Introduction to Auto-BG & Board Game Data

What is Auto-BG?

How does a board game evolve from an idea to physically sitting on your table?

This application attempts to augment one step, early in that journey, when disparate ideas combine into a holistic concept. Interpreting mechanical and descriptive tags to translate a game profile to machine-readable text, Auto-BoardGame (Auto-BG) uses a custom pipeline of GPT3 and T5 models to create a new description and proposed titles for a game that doesn't exist today. These descriptions support designers-to-be as alternatives to current concepts, seeds for future concepts, or any user as, hopefully, an entertaining thought experiment.

While general large language models (LLM) such as ChatGPT solve this use case by generating coherent descriptions from sentence-formed prompts, we believe this design niche would be better served by a dedicated application leveraging domain-specific transfer learning and granular control through an extensive tag-prompt framework.

Before digging into the process of generating a board game concept, let's ground some key terms:

- *Mechanical* - The gameplay mechanics which, together, create a distinct ruleset for a given board game. Represented here by a class of tags each capturing individual mechanics such as "Worker Placement" or "Set Collection".
    - *Cooperative* - Our application, and the dataset, singles out this mechanic as important enough to have its own tag class. As an alternative to the domain-default of *Competitive* gaming, *Cooperative* represents both an individual mechanic and a fundamental design paradigm shift.

- *Descriptive* - The narrative, production design, or community-relation elements of a game. This can include genre, categorical niches, or any relational tag not captured by the class of mechanical tags described above.

Understanding the Data

To train these models, we utilized a processed dataset of 20,769 ranked board games scraped from the board game forum BoardGameGeek.com[1](BGG). Each game includes its respective name, description, and 3,800 feature tags separated into five classes including Cooperative, Game Type, Category, Mechanic, and Family; when you select tags within Auto-BG, you choose and assign them within these distinct classes. To be ranked, games must have received a lifetime minimum of thirty user ratings; non-standalone expansion and compilation games have also been removed to reduce replication in the training data.

As previous studies[2] identified a disproportionate bias toward english language descriptions, our approach used a soft pass to remove games identified as having non-english descriptions using langdetect[3]. We also implemented a check on all remaining titles to remove any that used non-english characters. In total, this purged 1,423 or ~6.4% of the data.

The training data powering Auto-BG was originally scraped via the BGG XML API by Markus Shepard on GitLab[4] with derived features inferred from The Impact of Crowdfunding on Board Games (2022)[5] by N. Canu, A. Oldenkamp, and J. Ellis & Deconstructing Game Design (2022)[2] by N. Canu, K. Chen, and R. Shetty. That work, and Auto-BG, are licensed under CC BY-NC-SA 2.0. Our GitHub repository includes a python script collating the processing steps required to turn a raw scraper output file into the final data constructs required to run an instance of Auto-BG. All source data remains the property of its respective owner and all licensing conditions are respected within Auto-BG.

Ethical Considerations

As a chaotician once said “your scientists were so preoccupied with whether or not they could, they didn't stop to think if they should”[6] - but that’s not what we’re going to do. So before we continue, a few notes regarding potential biases and concerns when training or using instances of Auto-BG.

First and foremost, Auto-BG inherits critical biases from the base GPT3 Curie model. Extensive work exists documenting the model family’s biases related to race[7], gender[7,8], religion[9], and abstracted moral frameworks on right and wrong[10]. A key vector for producing biased output from these models comes from the conditioning context (i.e. user input) which prompts a response[11]; while these inherited biases can’t be eliminated, Auto-BG attempts to reduce their impact on output text through strictly controlling the conditioning context by relating unknown input to pre-approved keys.

Additionally, Auto-BG inherits an additional degree of bias in generating gender markers within descriptive text from the BGG training data. Observe an example of this in figure 1 - comparing the non-exclusive normalized per word appearance rate of "He" and "She" as markers in descriptions. For visibility the y-axis of BGG data has been limited to 100 entries, ~18k and ~20k descriptions use neither male nor female markers respectively.

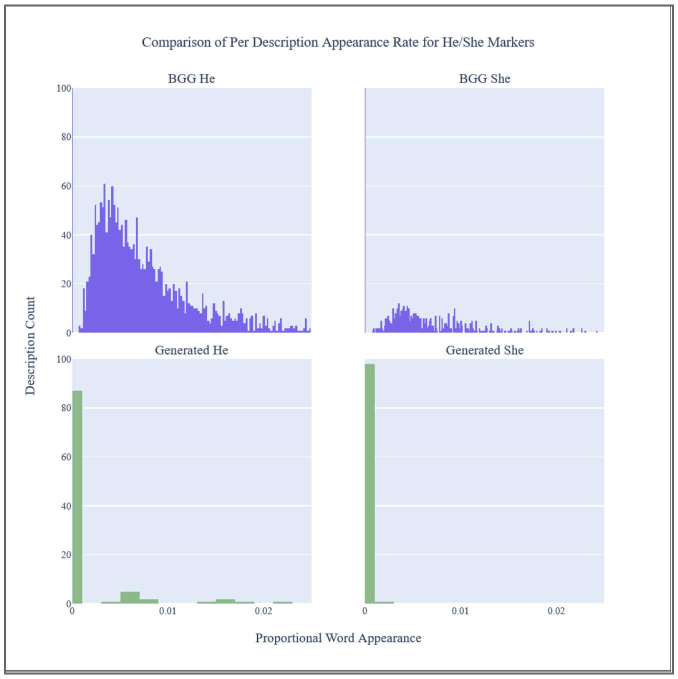


Fig 1. - Comparison of He/She markers in BGG vs generated text

While the vast majority of text includes no gender markers, those that do disproportionately weigh toward male markers. Similar right-skewed rate distributions also emphasize the disparity in appearance count - male markers aren't used more frequently within their text but appear more frequently. This transfers through Auto-BG's pipeline resulting in output that increases disparity when merged with existing GPT3 data.

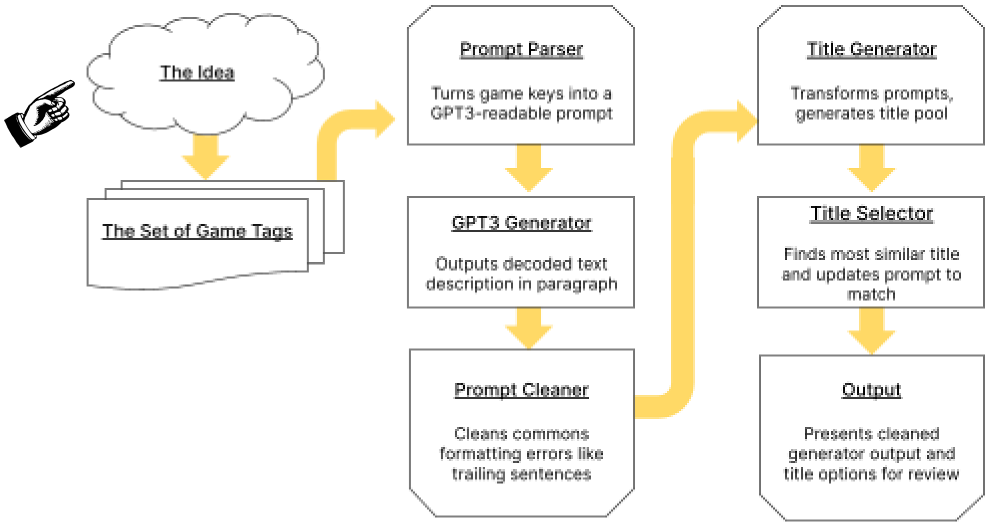
Though outside the feasible scope of this project, future iterations of Auto-BG should include coreference cleaning on the training input to minimize inherited bias; this approach would require a robust review of the source data through the framework of coreference resolution to avoid further harm by making poor inferences when resolving references[12].

Beyond these issues, with any LLM-based project, we have to ask - are we potentially harming human actors by deploying this tool? As we build Auto-BG in 2023, discussion of prospective gains and potential dangers of LLMs has broken out of the data science domain to be a growing global conversation. Academics wrestle with the scope and limitations of GPT models[13] while crossover publications debate topics including the merits of ChatGPT as an author[14] and which professions AI models might replace in the near future[15].

With these concerns in mind, we designed Auto-BG as a strictly augmentative tool - building a board game takes more than a conceptual description and, while Auto-BG may suggest rule frameworks, it can't generate any production elements such as full rulesets or art required to take the game from concept to table. In addition, we strongly encourage use of Auto-BG as a starting point, not a ground truth generation; its underlying models make mistakes that a human actor might not. While we have implemented catches to discourage Auto-BG from replicating existing board games, its GPT3 trained base model has an extensive knowledge of gaming in general. We recommend reviewing output text thoroughly before committing to your new board game being titled *Age of Empires II: Age of Kings* or *Everquest*©.

The Pipeline - Bird's Eye View

Auto-BG relies on multiple LLMs to generate a full game concept from your initial input. The entire process, from turning your tags into a prompt to presenting your finished concept, is embedded in a pipeline, a framework of related code running sequentially, that enables these distinct steps to work in unison. It all starts with an idea.


The Idea
The Set of Game Tags
Prompt Parser
Turns game keys into a GPT3-readable prompt
GPT3 Generator
Outputs decoded text description in paragraph
Prompt Cleaner
Cleans commons formatting errors like trailing sentences
Title Generator
Transforms prompts, generates title pool
Title Selector
Finds most similar title and updates prompt to match
Output
Presents cleaned generator output and title options for review

1. The most complex participant in Auto-BG's pipeline (that's you) translates elements of an abstract idea for a game into defined tags through the Auto-BG interface.

2. An input parser collates tags, turning them into a game vector (a binary, or one-hot, representation of a game where all selected tags are valued at 1). This vector activates internal keys representing each tag selected in step 1 and passes them as a prompt to Auto-BG's GPT3 model.

3. A generator function calls our model through the OpenAI API, returning a response using selected parameters for the input. This response is decoded and lightly cleaned for readability.

4. The cleaned output passes downstream to a fine-tuned t5 sequence-to-sequence model; it uses the output as input to generate potential candidate titles.

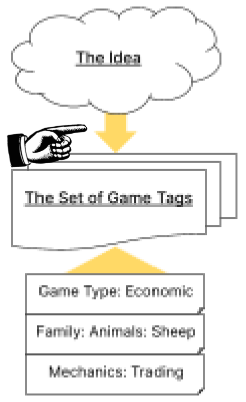
5. Because the GPT3 model learned that games should include a title, it may generate an artificial title in the output. To evaluate this, the title generator runs once and strips any identified candidate titles for a placeholder before running again.

6. A title selector rejoins the initial and secondary generation pools, removes duplicates, validates against existing game titles, then scores cosine similarity of each title with its reference input.
7. The Auto-BG user interface returns the generated prompt with its highest-scoring title. A higher-order function in the streamlit app access and updates titles & descriptions as you cycle through them for review.

At the end of the pipeline, Auto-BG has produced a brand-new game concept customized by your selected tags. To understand how Auto-BG achieves this, let's go through each step in detail. We'll also examine how an example prompt transforms at each step using game tags for the seminal title Catan.

The Inner Workings of Auto-BG

Designing The User Experience



We know not all readers come to Auto-BG with the same background, so if you want to skip around in the pipeline, look for this handy key:

Sections primarily including theory, methodology, design, or anything else on the "why" of Auto-BG

Sections primarily covering what happens under-the-hood in Auto-BG - all "whats", all the time* (*all the time not strictly guaranteed)

To develop the overall design for Auto-BG, we asked ourselves - who would be using this tool and what would they want to gain from it? This question guided the structure of Auto-BGs user interface and, by extension, our approach to integrating the various processing steps. In scoping the interface, we identified three core personas of potential users for Auto-BG, each of which contributed to our holistic design philosophy.

- The Developer needs a utility tool to augment designing board games. As a practical consideration, this user probably doesn't care about the machine learning (ML) fundamentals of Auto-BG; knowing more about how Auto-BG works benefits them only so far as it improves their efficiency at aligning tags with the desired output.

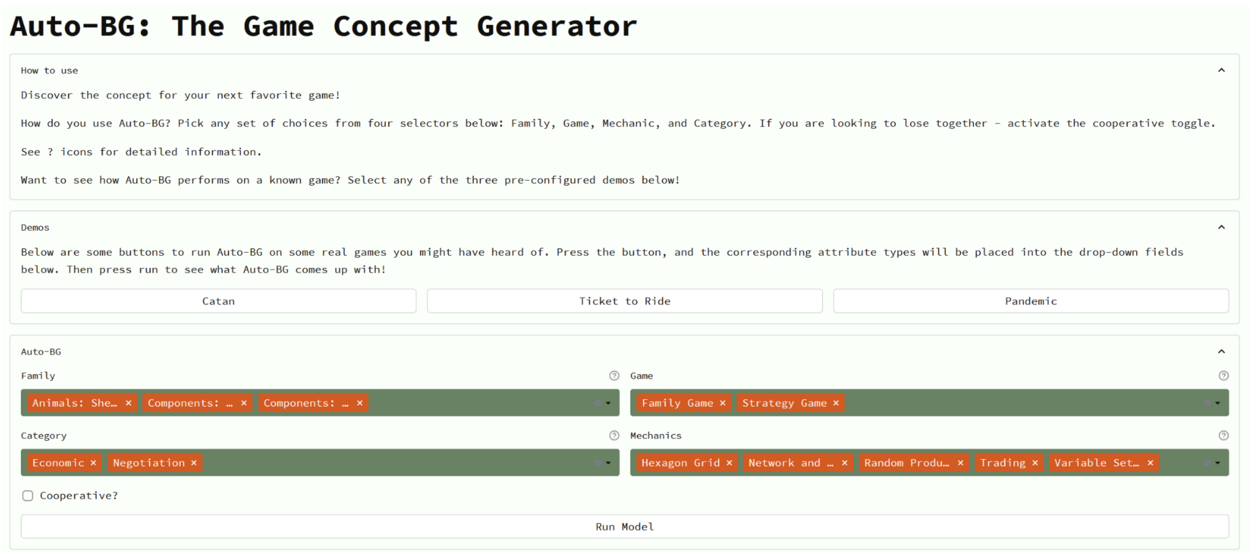
- The Experimenter wants to play with Auto-BG - they fundamentally come to the app to test its limits. While ML may interest them from a lay-perspective, they're here for a laugh and the technical aspects need to stay out of the way. We anticipate a typical user Auto-BG to fit this persona with varying degrees of the Developer mixed in.

- Finally, the Data Scientist seeks to understand Auto-BG as a natural language application. Whether for the knowledge itself, applying the techniques to their own application, or learning more about the underlying data, this user wants insight into what's happening inside Auto-BG. While this user inherits some features of both previous personas, we anticipate a unique input profile with higher diversity and a more rigorous relationship with output evaluation.

In developing wireframes for the Auto-BG interface, we considered how to balance the needs of each user persona. The Developer contributed collapsable information forms with limited guideline tooltips attached to each field that could be referenced quickly in real-time. Alternatively, an Experimenter with limited board game experience may need a more thorough demo; to accommodate this, we embedded pre-filled example games based on the tags for popular titles *Catan*, *Ticket to Ride*, and *Pandemic*.

The Data Scientist proved the most challenging persona to support without severe information bloat. In this case, we chose to add an additional page within Auto-BG that includes a compact version of this blog with extended data available through our GitHub repository.

Iterative testing of the wireframe with feedback from friends and family led to a responsive interface that meets these needs while seamlessly integrating the backend model pipeline. See an example of it below then follow along to learn in-depth about how Auto-BG generates a game concept.



Auto-BG: The Game Concept Generator

How to use

Discover the concept for your next favorite game!

How do you use Auto-BG? Pick any set of choices from four selectors below: Family, Game, Mechanic, and Category. If you are looking to lose together - activate the cooperative toggle.

See ? icons for detailed information.

Want to see how Auto-BG performs on a known game? Select any of the three pre-configured demos below!

Demos

Below are some buttons to run Auto-BG on some real games you might have heard of. Press the button, and the corresponding attribute types will be placed into the drop-down fields below. Then press run to see what Auto-BG comes up with!

Auto-BG

Family: Animals: She… ×, Components: … ×, Components: … ×

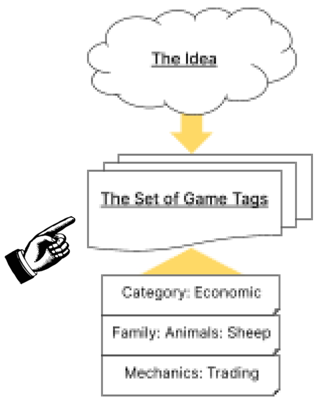
Game: Family Game ×, Strategy Game ×

Category: Economic ×, Negotiation ×

Mechanics: Hexagon Grid ×, Network and … ×, Random Produ… ×, Trading ×, Variable Set… ×

☐ Cooperative?

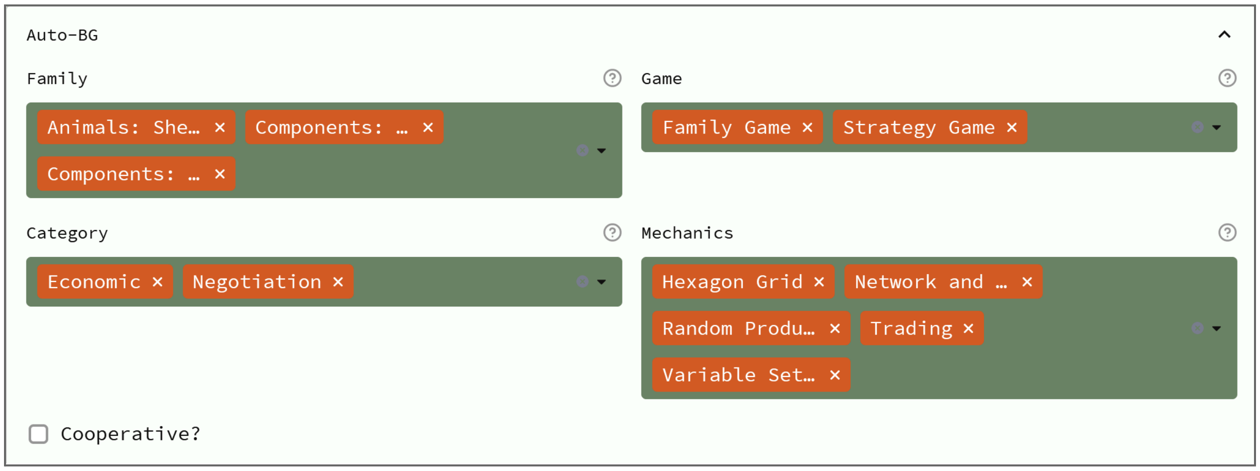
Interpreting User Input



How do individual tags add up to a game profile?

A complex series of transformations from input to generation requires understanding both how a human reader interacts with and interprets game tags but also how the GPT3 LLM at the heart of Auto-BG could learn and understand them.

Auto-BG inherits tag classes from BoardGameGeek; on any game page you'll find a collection of type, category, mechanic, and family tags on a sidebar. Auto-BG 1.0 explicitly focuses on these four primary classes as their text-only format translates coherently to a GPT3 prompt training structure, universally applies to an unknown game without changing the ground truth tag pool, and doesn't rely on post-design information including user recommendations or publisher data. Auto-BG 1.0 does exclude some high value features such as player count or play time due to the added complexity of converting them to a natural language format.


Auto-BG

Family: Animals: She… × Components: … × Components: … ×

Game: Family Game × Strategy Game ×

Category: Economic × Negotiation ×

Mechanics: Hexagon Grid × Network and … × Random Produ… × Trading × Variable Set… ×

☐ Cooperative?

Catan - Game Tags in Auto-BG

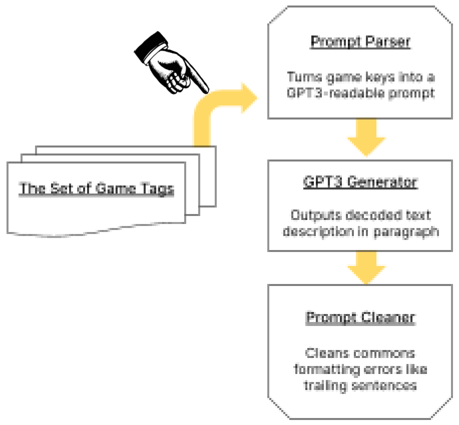
What happens when you choose tags in Auto-BG?

Behind the scenes, each tag has a hidden class prefix attached that tracks its affiliation throughout the pipeline. When you select a new tag, even if it's semantically similar to an overlapping tag in a different class, Auto-BG remembers that it belongs in mechanics, family, etc.. With this approach, Auto-BG, downstream, effectively handles unknown inputs by scanning the associated class and matching an unknown tag to its closest existing semantic relative.

Grouping your selected tags, Auto-BG creates an approximate profile of a new game in the same format as any existing BGG entry. When generating a new concept, Auto-BG attempts to infer additional features based on tags provided. It may include features such as player count, age rating, and play time in your concept if they frequently appear in descriptions for games with those same tags.

By changing even a single tag, Auto-BG will update and generate a different concept incorporating that new information. With this sensitivity to input selection, the user interface needed coherent tutorials and targeted maximum tag limits to guide users. To find reasonable outlier limits on tag selection without impeding the user experience, we analyzed the distribution of each tag class and set a limit at two standard deviations above the mean rounded to the nearest whole number. This provides a generous total tag space while minimizing the impact of noisy input on generation.

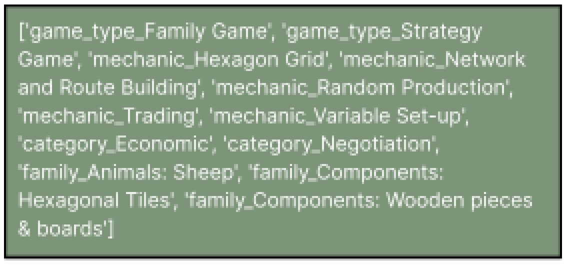
How does Auto-BG interpret your game profile?


The Set of Game Tags
Prompt Parser
Turns game keys into a GPT3-readable prompt
GPT3 Generator
Outputs decoded text description in paragraph
Prompt Cleaner
Cleans commons formatting errors like trailing sentences

Once you've created a new game profile, Auto-BG translates it into a formatted text prompt the GPT3 LLM understands. This conversion is performed following OpenAI's recommended best practices for conditional generation prompts[16,17] and results in a text prompt version of your tag list with each tag period stopped.

Fine-tune training taught Auto-BG that this specific prompt format should translate to a descriptive paragraph of its associated tags. To create the prompt, Auto-BG passes tags through an input manager python class that performs several steps in sequence before sending a refined prompt to the generator.

The input manager establishes a ground truth key dictionary of all existing tags - this format allows for iterative updating from future data as new tags appear on BGG. This python dictionary structure includes a key for every tag with its equivalent value set to 0, tracking that it does not appear. Eventually these change for your selected tags to 1, telling Auto-BG those tag keys appear in your game design and should be added to the prompt. However, before it does this, it tries to update unknown inputs.

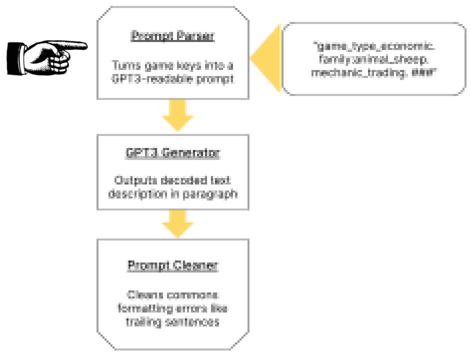

```
['game_type_Family Game', 'game_type_Strategy Game', 'mechanic_Hexagon Grid', 'mechanic_Network and Route Building', 'mechanic_Random Production', 'mechanic_Trading', 'mechanic_Variable Set-up', 'category_Economic', 'category_Negotiation', 'family_Animals: Sheep', 'family_Components: Hexagonal Tiles', 'family_Components: Wooden pieces & boards']
```
Catan - Prefix-Formatted Tags

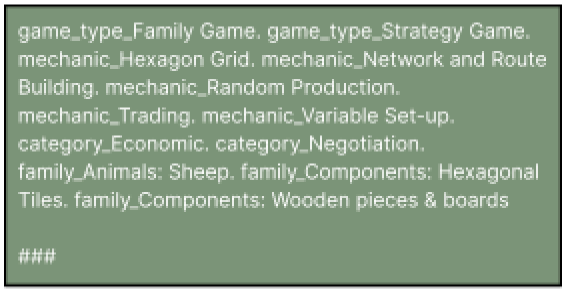
Each tag not matching an existing entry in the key dictionary moves to a holding pool until all input tags are assessed. With all unknowns marked, the input manager implements a within-class comparison between each unknown tag and all known tags sharing its prefix.

Auto-BG estimates semantic relevance between tags through token cosine similarity[18], comparing an unknown tag to each in-class match candidate with SpaCy's token similarity method[19]. After evaluating all candidates, the highest scoring key match for each unknown tag is activated and added to the input prompt.

Auto-BG's description generator inherits a characteristic sensitivity to prompt design of LLMs; prompt design often requires significant human effort and many approaches have been suggested to address its challenges. Our design philosophy focused on improving quality and control of prompts with goals of reliability, creativity, and coherence to input - specifically inspired by principles of metaprompting[20] and internal prompt generation within the LLM[21]. Reducing human-judgment interactions with the input prompt was a significant blocker to developing Auto-BG; to work around this, it leverages existing human tagging work already performed on the source data to provide the generator with a controllable abstract prompt instead of potential complexities of natural language sentence-structured prompts.

All pooled tags convert to a period-stopped sentence which splits the difference between the categorical structure of the training data and natural language base of GPT3. A generation function will send this prompt to our remote model through the OpenAI API and return a matching description.


Prompt Parser
Turns game keys into a GPT3-readable prompt
"game_type_economic. family:animal_sheep. mechanic_trading. ###"
GPT3 Generator
Outputs decoded text description in paragraph
Prompt Cleaner
Cleans commons formatting errors like trailing sentences


game_type_Family Game. game_type_Strategy Game. mechanic_Hexagon Grid. mechanic_Network and Route Building. mechanic_Random Production. mechanic_Trading. mechanic_Variable Set-up. category_Economic. category_Negotiation. family_Animals: Sheep. family_Components: Hexagonal Tiles. family_Components: Wooden pieces & boards

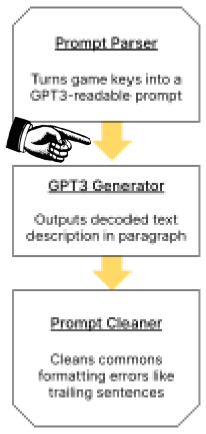
###

Catan - Final Input Prompt

Generating a Description

Selecting a Model

Prioritizing reliability, creativity, and coherence in generated descriptions, we evaluated the suitability of several LLMs for the core step of Auto-BG. Generally text-generation tasks


Prompt Parser
Turns game keys into a GPT3-readable prompt
GPT3 Generator
Outputs decoded text description in paragraph
Prompt Cleaner
Cleans commons formatting errors like trailing sentences

require a large, diverse training corpus and extensive parameter fine-tuning for specific tasks. With a limited corpus size for BGG, we determined that any application would necessitate building on top of an existing model. After reviewing multiple LLMs including Text-to-Text Transfer Transformer (T5 - eventually used in title generation), Bidirectional Encoder Representations from Transformers (BERT), and Generative Pre-trained Transformers (GPT), we determined that GPT3, the most recent iteration available to fine-tune, would best fit our goals for Auto-BG.

A causal language model released by OpenAI in 2020, GPT3 has nearly 175 billion parameters. With expansive pre-trained corpus, it estimates the probability of a sequence of tokens occurring in text given initial prompt, also referred to as a conditioning context. This approach, conditional text generation, results in an output sequence conditioned on the prompt input, being implicitly controlled by the prompt structure. In practice, the model predicts the next word in each sequence of text based on the context of previous words. using neural network architecture as a transformer. Unlike other transformer models with an encoder and decoder, each with multiple layers of self-attention and feed-forward neural networks, GPT-3's only utilizes the decoder structure.

The decoder leverages encoded text representations from its corpus and generates an output sequence. At each layer, the model attends to the encoded representation and previous tokens in the output to generate a new token. By compiling these, the final output of the decoder becomes a complete output sequence. While a purely deterministic GPT3 model would generate the same output given an identical input, Auto-BG introduces a degree of chaos to the generation along with two regularization penalties to encourage unique outputs given the same conditioning context.

Training Methodology

In machine learning, transfer learning applies additional training to a model originally trained on one task or domain and teaches it a different task. This approach has proven effective for many natural language processing (NLP) tasks, allowing new models to leverage pre-existing knowledge of language and syntax. However, models fine-tuned using transfer learning may be limited by the quality of the pre-trained model and require substantial amounts of labeled data to fine-tune effectively. To mitigate this, implementing transfer learning training requires careful consideration of how you perform-completion pairs for training.

Using structured prompt-completion pairs formatted as a JSONL document, you can fine-tune a new model to learn and perform a specific task. As a highly tailored approach, this often achieves better performance with less labeled data. Additionally, this form of prompt engineering allows for greater control over model output with less ambiguity in the generation steps. For Auto-BG, a Curie-tier GPT3 model was fine-tuned on training prompts consisting of all active feature keys for each game item transformed into a string with each key period stopped; the model learned that each of these key prompts should result in the matching description as an output.

While less cost-intensive than training a new model, fine-tuning a GPT3 model remains relatively expensive and time-consuming. To address this, we began with a thorough review of the literature on LLM tuning as well as OpenAI's own recommendations for conditional generation models. OpenAI provides detailed training metrics via a downloadable csv file for each model which includes five critical metrics: total tokens recognized by the model at each step (elapsed tokens); examples the model has seen relative to batch size (elapsed examples); loss on training batch (training loss); percentage of exact completions predicted (a sequence of tokens matching the previous prompt completion examples exactly relative to batch size); and percentage of tokens in the training batch correctly predicted[16]. Illustrated in figure 2, a median training loss score of 0.77 (IQR of around 0.245) and median training token accuracy of 0.464 (IQR of around 0.017) with training sequence accuracy score consistently at 0 indicates Auto-BG's text model produced results similar to corpus descriptions without direct reiteration.

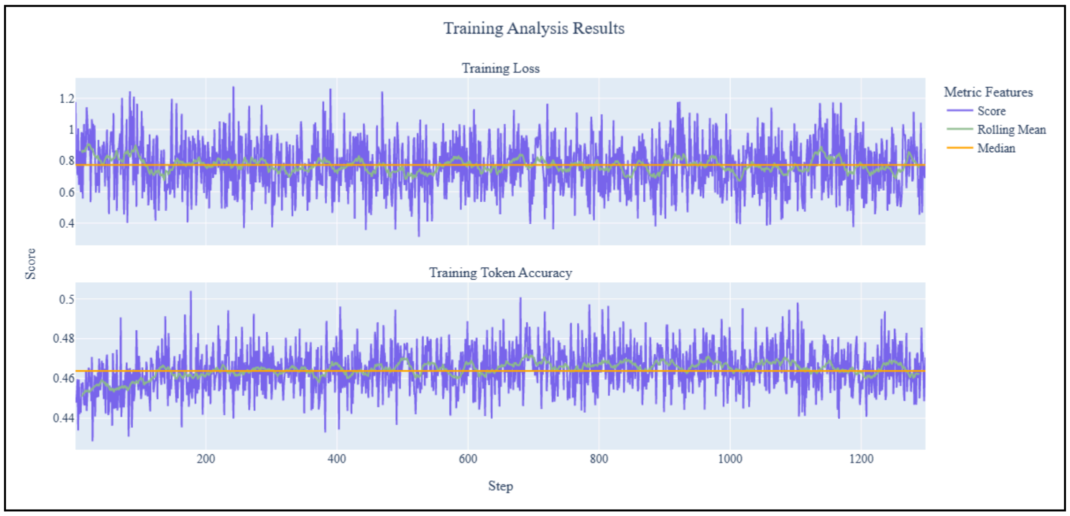


Fig 2. - Training Loss and Training Token Accuracy

We ultimately trained the implemented model using a lowered learning rate of 0.02, extending training time but increasing information gain at each step; to balance this, OpenAI suggests conditional text models use less training epochs which we chose to follow, training for two instead of the baseline of three. With limited ability to iterate this model, we focused efforts on refining generation parameters to improve performance.

To assess performance, we identified three key generation parameters impacting output quality - temperature, presence penalty, and frequency penalty. These control randomness vs deterministic operation, penalize tokens that already appear in the text, and penalize tokens for their current in-text frequency respectively[22]; taken collectively they directly affect the semantic coherence and ability of the generator to create unique outputs that diverge from the training data.

Using a grid search approach to identify the outer bounds of each value where generation coherence began to degrade, we iteratively narrowed parameter profiles to find optimal candidates. Each parameter profile was scored using a panel of seven metrics including precision, recall, and F1 from BERT Score[23], uni-, bi-gram, and longest common sequence rouge[24], along with BLEURT[25].

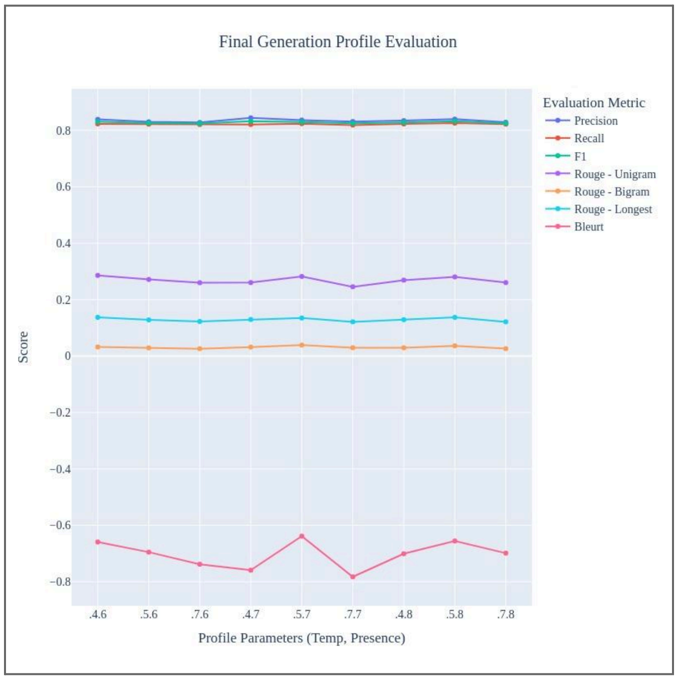


Fig 3. - Final Parameter Profile Evaluation Metrics

This final metric provided the greatest insight into model performance as it is primarily a comparative metric assessing semantic similarity to a reference text but also broadly estimates relative fluency and coherence. Figure 3 shows the breakdown of relative scores; all metrics except Bleurt are bound from 0-1 while it can fall between -1-1. In its implemented version, Auto-BG uses profiles with a temperature and presence penalty of .5/.7, .4/.6, and .5/.8 respectively.

With a final set of three generation parameter profiles locked in, we needed to confirm that the text generator hadn't simply memorized the training data. Critically, in Auto-BG, there are no guardrails to prevent a user from inputting a tag prompt matching a unique value in the training set - in many cases it could be key that the generator output doesn't replicate training data.

To test this, we utilized the same metrics panel against each generation profile for a random sample of one-hundred games from the training set. By evaluating performance, we believe that, in the case outlined above, Auto-BG would successfully generate a distinct output. Figure 4 visualizes these results across each generation profile per sample game to assess outliers.

While the Precision, Recall, and F1 scores typically range between .7 and .9 for all profiles, we found that Rouge scores were reasonably dissimilar, below a .5 threshold. For BLEURT we can interpret 1 as a perfect replication with 0 as purely random, scores below 0 semantically move away from randomness in a way that's consistently dissimilar to the reference. Given BLEURT scores below 0 and moderately low Rouge scores, it is unlikely the generator has memorized training text to prompt relationships in a way that causes it to deterministically replicate the training data at generation. At the same time, high Pre/Rec/F1 scores suggest token-level similarity, indicating the generator performs as intended by learning the linguistic structures of a prompt without being semantically identical.

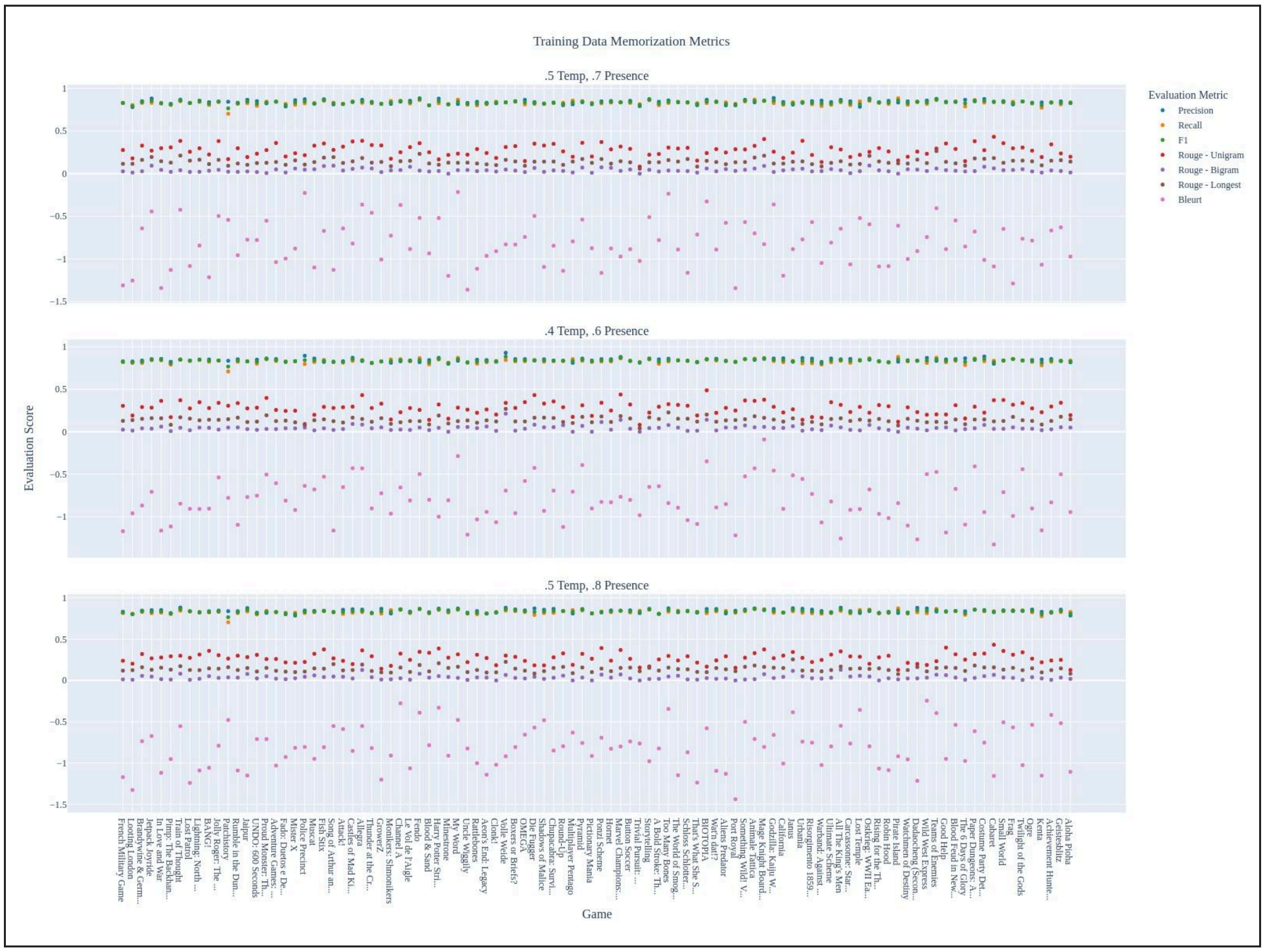
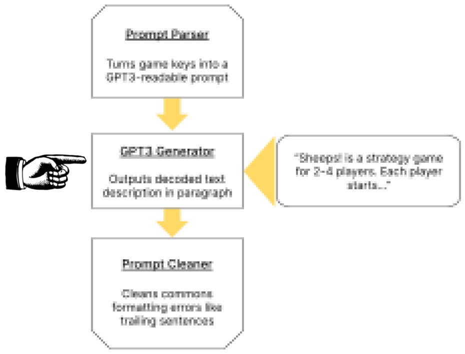


Fig 4. - Training Data Evaluation Metrics

Output Generation

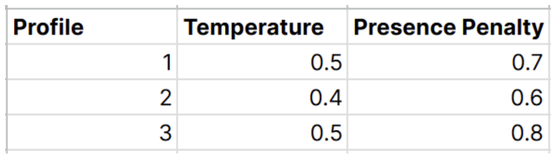

Prompt Parser
Turns game keys into a GPT3-readable prompt
GPT3 Generator
Outputs decoded text description in paragraph
"Sheepsl is a strategy game for 2-4 players. Each player starts..."
Prompt Cleaner
Cleans commons formatting errors like trailing sentences

As output varies greatly based on not only the input but the generation parameters, Auto-BG uses a multi-output approach in both models that reduces performance but prioritizes the likelihood of a user receiving a satisfactory output. To support each of our user personas, we believe the net harm of returning less viable output pairs is outweighed by the value of potentially generating even a single output that satisfies their task-specific needs.

Once Auto-BG collates the period-stopped input prompt, it passes this through an API call to the remote GPT3 model three times, creating a unique output for each of the parameter profiles. The relative difference in training data plays a major role in why Auto-BG utilizes this approach to generation over other implementations. As BGG only collates existing game data, there isn't per feature standardization and the robustness of training examples for a given tag varies.

The downstream impact of this while generating is that prompts have diverse momentum with high-volume tags requiring increased temperatures to break away from training information. Alternatively, low-volume tags may fail to achieve fluency or coherence to the selected tags at high temperatures.

In the output step, frequency penalties remain fixed at 0.5 while the other key parameters shift for each profile. In review, we found that all other values of frequency penalty eventually resulted in catastrophic failure for text fluency with increasing text repetition or a loss of topical focus depending on the direction.

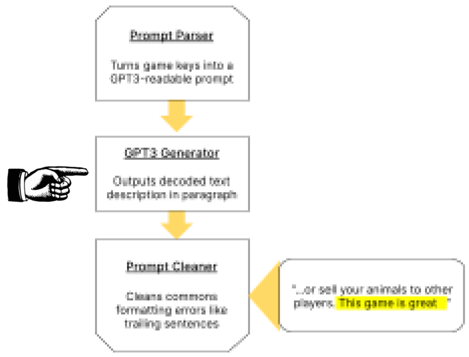


Profile	Temperature	Presence Penalty
1	0.5	0.7
2	0.4	0.6
3	0.5	0.8

Parameter Profile Reference

Although each of the parameters changes only slightly, this results in visibly varied output based on both human-feedback and observed training results. Beginning with a moderate value for each parameter, the generator cycles through a more deterministic and random profile before returning three distinct descriptions.

Output Processing


Prompt Parser
Turns game keys into a GPT3-readable prompt
GPT3 Generator
Outputs decoded text description in paragraph
Prompt Cleaner
Cleans commons formatting errors like trailing sentences
"...or sell your animals to other players. This game is great. "

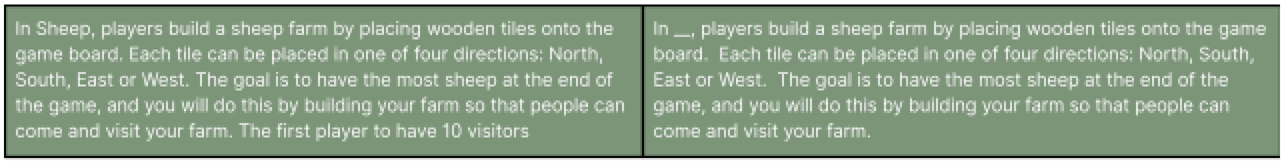
While Auto-BGs baseline output performs adequately on metrics based assessments, we want to add a few extra cleaning steps - one generic to conditional generation and two domain-specific.

These parameter profiles minimize the frequency of semantic irregularities like trailing sentences or token limit hard stops but both can still appear in text. To filter these the generator checks the final non-whitespace character, if it detects anything other than a period, question mark, or exclamation point, it truncates the last sentence to the previous punctuation stop. Despite occasionally flagging false positives, in testing this reduced the possibility of outputs ending unexpectedly and instead reaching a semantically-coherent conclusion.

Two acute issues affect Auto-BG's text generation, both arising from the BGG training data. The first of these, inadvertent title generation in the body text, is managed downstream so we'll cover it there. Addressing the other proved to be one of the more challenging blockers to clear in designing Auto-BG.

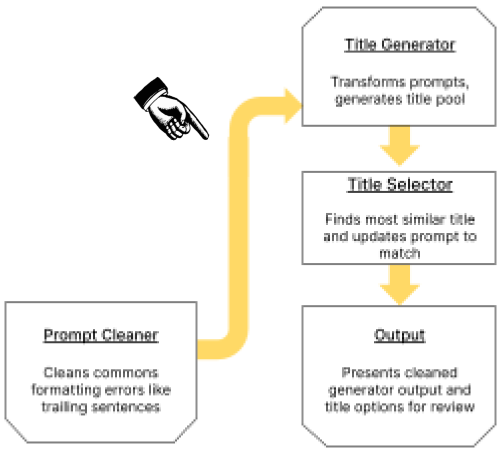
In training the text model learned that game descriptions may reference their designer, artist, or publisher. Additionally, they can include other games associated with that person or entity. To make this issue more difficult to prune, Auto-BG knows to include these features but lacks comprehension to understand coreferences between them and a related game; this results in two equally catastrophic scenarios where a person who doesn't exist is paired to a real game or an actual person is matched to a game they had no involvement in. With the resources available to it, Auto-BG uses a relatively naive solution to the problem but one that has shown minimal impact on overall output. Using keyphrases such as "from the designer of", a cleaner function splits and searches each sentence, removing those that contain the keyphrases entirely.

With these processing steps complete, Auto-BG sends the outputs downstream to be evaluated and paired with their respective titles. The graphic below shows a basic example of how an initial output is transformed and cleaned throughout the pipeline, also including title identification.



In Sheep, players build a sheep farm by placing wooden tiles onto the game board. Each tile can be placed in one of four directions: North, South, East or West. The goal is to have the most sheep at the end of the game, and you will do this by building your farm so that people can come and visit your farm. The first player to have 10 visitors	In __, players build a sheep farm by placing wooden tiles onto the game board. Each tile can be placed in one of four directions: North, South, East or West. The goal is to have the most sheep at the end of the game, and you will do this by building your farm so that people can come and visit your farm.
Catan Prompt - Initial Generation	*Catan Prompt - After All Cleaning*

Fitting a Title


Title Generator
Transforms prompts, generates title pool
Title Selector
Finds most similar title and updates prompt to match
Output
Presents cleaned generator output and title options for review
Prompt Cleaner
Cleans commons formatting errors like trailing sentences

For the final touch completing the game concept, Auto-BG provides a selection of titles fit to the new description. In scoping the title generation model, we asked "what defines a good title?" and realized it depends on the domain and specific product. If a sense of tension matches a thriller novel[26] and positive associations enhance consumer products[27], what best fits a given board game? To create titles that align with both their description and domain trends as a whole, Auto-BG needed a model to interpret the unique relationship between titles and description in the training data.

What model does Auto-BG use to make titles?

While the text generator uses a prompt-based GPT paradigm, title generation implements a different approach inside Auto-BG. T5 models leverage a NLP framework known as sequence-to-sequence or text-to-text[28]; at their core, they're an encoder-decoder model which treats all NLP operations as a translation task. Given input text with one semantic structure, it outputs a predicted target in a different structure that best fits as a "translation".

Instead of relying on the relatively limited corpus of 20,000 BGG descriptions alone, likely inadequate for fresh training of a generative NLP model, Auto-BG implements two-stages of upstream transfer learning through HuggingFace's Transformer library. In machine learning, transfer learning applies additional training to a model originally trained on one task or domain and teaches it a different task. This final implementation extends a model[29] already fine-tuned once on an additional 500,000 news articles from t5-base[30].

To select the best generator for Auto-BG, the team fine-tuned multiple iterations on both t5-base and headline-trained models. The final round of training utilized a higher learning rate of .001 while introducing a weight decay of 0.01 in the optimizer; work by Tay, et al.[31] on scaling and fine-tune training for transformers guided narrowing these parameters as the compute cost to iterate models was substantial.

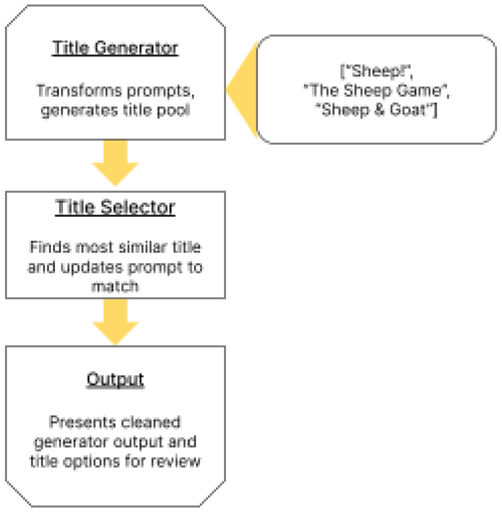
In assessing these models, we recognized the need for distinct metrics from the generator implementation. As game titles in the corpus tended toward brevity, typically less than three words, we chose to focus on Word Mover's Distance[32](WMD), a metric that excels at evaluating semantic relationships between a generated and reference title at the per-word scope.

WMD uses pre-trained word embeddings, commonly word2vec in Python, to calculate semantic dissimilarity between two pieces of text through the total cost of movement to transfer from one embedded space to the other. In practice, this results in a score that reasonably represents the semantic connection of two titles even when they share no text in common.

Across a grid search of beam counts with varied diversity & repetition penalties, the base and headline models trained with the modified parameters consistently outperformed their baseline counterparts. While the base-t5 model produced stronger WMD scores, indicating a smaller semantic distance to reference in its titles, we ultimately chose to implement the headline-trained model within Auto-BG; human review revealed that the headline model produced less deterministic titles with increased diversity in each multi-output pool of titles, better supporting our downstream tasks.

Generating New Titles

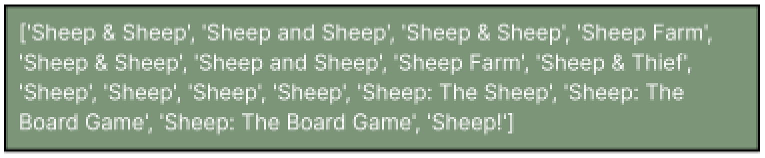
Several challenges arose from syncing two models with no inherent awareness of one another. In particular, the description generator also learned that games often include their title in the body text. To avoid discrepancies in the final output, Auto-BG uses a two-pass generation function to identify and strip titles from the text. This means that when a new description flows to the title model, it has to handle both an additional generation step and reconfigure its input.

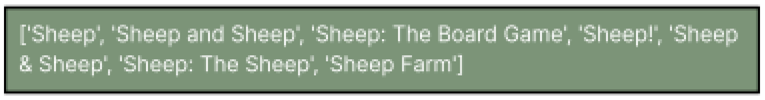

Title Generator
Transforms prompts, generates title pool
["Sheep!", "The Sheep Game", "Sheep & Goat"]
Title Selector
Finds most similar title and updates prompt to match
Output
Presents cleaned generator output and title options for review

As the source data includes an embedded title in 63% of game descriptions, the text model learned to include them but the title model also prioritizes identifying high-likelihood phrases to be titles in the body. Before any selections are made, Auto-BG leverages this implicit relationship between the models in a cleaning step. It performs title generation then uses a simple regular expression check against the candidate titles, replacing any identified in the description with a placeholder token.

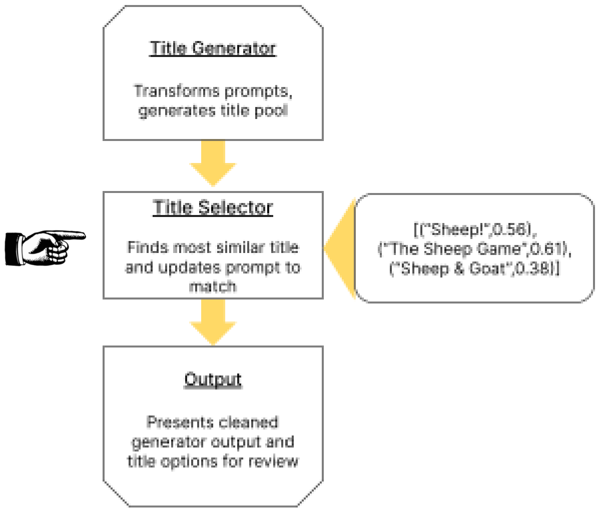
Despite removing the identified titles, Auto-BG retains this first pass generation - if the first model generated a more fitting title, we don't want to discard it. After hiding the placeholder a second set of titles generates, then both sets are joined for all downstream scoring which supports a large and diverse candidate pool. Since Auto-BG lets users select titles, it prefers retaining lower quality over potentially discarding a title the user would eventually select; while title scoring frequently bubbles up appropriate titles, it's not a perfect system and both Developer & Experimenter personas benefit more from diversity than streamlining for precision.

At this point, the total pool may include undesirable titles - duplicates, existing games, etc. Since the GPT3 model has underlying training data with an awareness of broad gaming topics, Auto-BG faces challenges with uniqueness both in existing BGG data and outside titles. Before sending this joined title pool downstream it strips entries found in the source data as well as a manually input set, then performs


```
['Sheep & Sheep', 'Sheep and Sheep', 'Sheep & Sheep', 'Sheep Farm', 'Sheep & Sheep', 'Sheep and Sheep', 'Sheep Farm', 'Sheep & Thief', 'Sheep', 'Sheep', 'Sheep', 'Sheep', 'Sheep: The Sheep', 'Sheep: The Board Game', 'Sheep: The Board Game', 'Sheep!']
```
Catan Prompt - Total Pool of Candidate Titles


```
['Sheep', 'Sheep and Sheep', 'Sheep: The Board Game', 'Sheep!', 'Sheep & Sheep', 'Sheep: The Sheep', 'Sheep Farm']
```
Catan Prompt - Titles After Cleaning

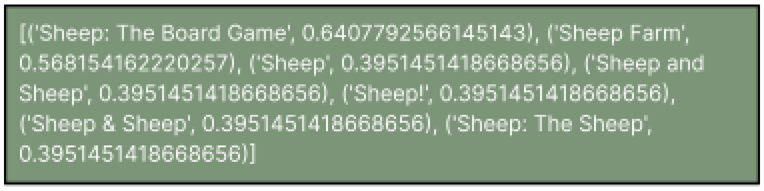
a case-insensitive check to remove duplicate titles. These steps act as a soft filter to encourage the model to pass a maximum unique pool to the final output.


Title Generator
Transforms prompts, generates title pool
Title Selector
Finds most similar title and updates prompt to match
[("Sheep!",0.56), ("The Sheep Game",0.61), ("Sheep & Goat",0.38)]
Output
Presents cleaned generator output and title options for review

Refining Titles & Updating Outputs

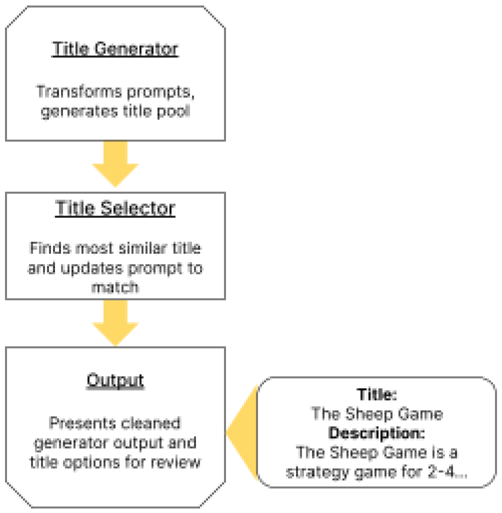
Given a processed candidate pool, Auto-BG tokenizes the generated description and each candidate title then uses SpaCy's *en_core_web_md*, an english language pipeline model, to evaluate token cosine similarity for each pair. As an evaluation metric, cosine similarity works well for this task as input and target lengths vary[18]. By evaluating word embeddings, we mitigate significant differences in magnitude.

Returned scores fall between 0 and 1, they estimate the semantic relevance of titles to their paired descriptions. As some tokens are unknown to SpaCy, the scorer assigns a random value between .3-.6 for sorting, a value spread approximating the first to third quartiles; since all titles remain available to users this is purely cosmetic for presentation order.


```
[('Sheep: The Board Game', 0.6407792566145143), ('Sheep Farm', 0.568154162220257), ('Sheep', 0.3951451418668656), ('Sheep and Sheep', 0.3951451418668656), ('Sheep!', 0.3951451418668656), ('Sheep & Sheep', 0.3951451418668656), ('Sheep: The Sheep', 0.3951451418668656)]
```
Catan Prompt - Scores Prior to Sorting

With scores associated, titles are sorted and stored in a nested dictionary with the placeholder tagged description. These dictionaries will be accessed post-generation and allow users to cycle between a description and its associated titles in real-time. A final step sets the display to the first description with its highest scoring title, replacing in-text references using a regular expression.

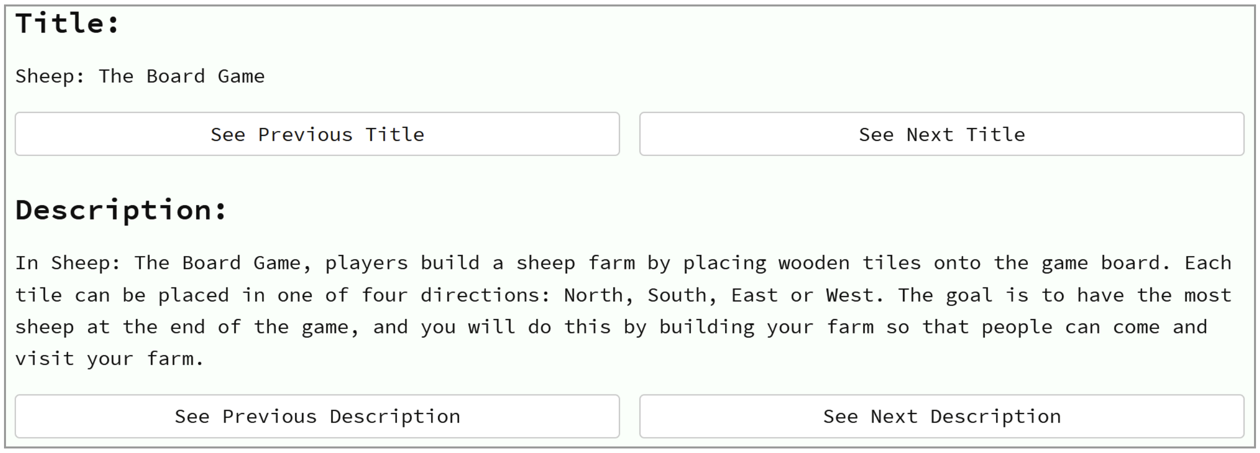
Controlling the Final Output


Title Generator
Transforms prompts, generates title pool
Title Selector
Finds most similar title and updates prompt to match
Output
Presents cleaned generator output and title options for review
Title:
The Sheep Game
Description:
The Sheep Game is a strategy game for 2-4...

With the display set to Auto-BG's baseline profile and its best scoring title, the user can now cycle through titles and descriptions. Behind-the-scenes, the user interface will seamlessly replace all title references in a given description text with the current title. A typical title pool provides two to five candidates for each generated description. Auto-BG tracks each of these using a simple key-value dictionary and a storage pair saved to the Streamlit session state.

To prevent generation calls spamming the OpenAI API, Auto-BG prevents users from immediately re-running the same tag prompt. Changing any tag value will allow generation to run again and, if you need to regenerate the same prompt, this can be accomplished by resetting the internal application cache.

At this point you will have a full set of descriptions and associated titles to review. Auto-BG offers a few additional ways to interact with the output - you can leave us general feedback by selecting from the sidebar or report an inappropriate title by selecting the report button at the bottom of the page. These prompts are license-free and can be used without attribution but we would appreciate a shout-out if you like what Auto-BG is doing!



Title:

Sheep: The Board Game

Description:

In Sheep: The Board Game, players build a sheep farm by placing wooden tiles onto the game board. Each tile can be placed in one of four directions: North, South, East or West. The goal is to have the most sheep at the end of the game, and you will do this by building your farm so that people can come and visit your farm.

Catan - Final Output

Takeaways from Auto-BG

A Different Approach to Generation

That was just a glimpse into the inner workings of Auto-BG's pipeline, our GitHub repository features the full code powering it. As for what this new approach brings to the table - Auto-BG focuses on scoping domain-specific tools; it takes a radically different approach to codifying and presenting individual components of a game design from general-purpose prompt models.

With over thirty-five hundred game tags in source data that powers Auto-BG, how could an average user, whether Developer, Experimenter, or Data Scientist, be able to identify, map their concept, and translate an interpretable prompt for a generator? Auto-BG mitigates access blockers by presenting an intuitive, searchable corpus of tags. The framework tying descriptive language to a categorical component already exists within BGG, leveraging it removes one step of cognitive load from users, freeing them to iterate organically in refining their game design.

And for users who understand mapping their design, turning that into a viable generator prompt may create further challenges. Common approaches to prompting in popular tools, such as ChatGPT, use conversational natural language formats, requiring additional expertise to prompt for desired outputs. The difficulty of prompt design remains a growing concern more generally, see the rapid increase of searches related to prompt engineering in 2023 alone[33], and Auto-BG streamlines this by removing direct input from the conditioning context. Our core philosophy revolves around refocusing user attention from the generator to their design and its needs - intentionally reducing awareness of Auto-BG as an active agent they're engaged with.

What Next?

We can't predict how Auto-BG will stand up to the rigors of public access but it shows promise at generating original concepts not simply memorized from existing data. At the same time, it exhibits core flaws of LLM generators: it misses, it hallucinates, and it occasionally produces potentially harmful content in spite of best efforts to corral it. Perhaps future iterations of Auto-BG, or a sibling application, can incorporate additional elements to mitigate these outstanding concerns based on the rapid iteration happening in LLMs.

From a practical standpoint, using the same data, we'd suggest these alternative approaches may prove fruitful. One addition Auto-BG 1.0 left on the cutting room floor is a game's nearest neighbors as a meta-feature - we provide scripts for our tested implementation of this on Github. We examined the value of aggregating jaccard similarity per feature class to create n-nearest neighbor communities, a useful resource in multiple downstream functions.

Additionally, we encourage exploring contrastive evaluation across multiple generations; Auto-BG currently outputs three concept profiles, allowing users to explore them, but alternatively could produce an n-set of generations from a single prompt. Scoring these against a community of games could improve performance and remove an additional layer of required user interaction. Within Auto-BG 1.0, using the OpenAI API framework, this proved cost-prohibitive but future implementations with more resources should consider this as a natural extension.

Future expansions of Auto-BG's core functionality should look toward information vectors intentionally excluded in this implementation. At a glance, these include incorporating additional languages or merging discarded feature classes with quantitative data to improve input diversity and precision.

As BGG disproportionally contains English-language descriptions, we recommend exploring data from other resources such as spielen.de[34]. While sites may not match the key structure of BGG, BGG includes documentation of alternative game titles in multiple languages; fuzzy text matching on this feature can reasonably join the training data with additional content. We suggest visiting the GitHub for The Impact of Crowdfunding on Board Games[5] for an example of how this approach was applied to merging BGG game items with Kickstarter data that shared no joint keys.

To properly allot development time for Auto-BG, the team made hard decisions about features to cut; these included a variety of quantitative features already embedded in the source data. Future iterations of Auto-BG will work towards incorporating features including age rating, player count, play time, and weight (a BGG specific metric approximating complexity). Auto-BG 1.0 currently infers these from input similarity to existing games which creates a major blocker in granular design generation. Among potential next steps, this represents the largest missing component from the original vision for Auto-BG and we anticipate that including it would significantly improve Auto-BG's ability to tune outputs to input specifications.

Despite the work to still be done, we believe Auto-BG represents a major step forward for bespoke data science applications in board gaming. The domain remains largely untapped relative to other cultural arts for both analysis and application, so we fully support any projects utilizing the data and framework of Auto-BG to engage with board gaming. For now we encourage everyone to get their hands dirty testing the limits of Auto-BG; experiment with niche designs, try your wildest ideas, and, most importantly, leave feedback in the app. We're excited to see what you create with Auto-BG!

Notes:

GitHub

See our GitHub for full code, embedded datasets, work statements, and additional appendices covering alternate approaches to building Auto-BG.

Licensing

Work Statements

N. Canu trained & implemented text and title models for Auto-BG, and led drafting and revision for this blog.

S. Capp led literature reviews, assisted with practical model research, visualizations, & UI components.

T. Druhot scoped and designed the Auto-BG UI, and implemented the integrated Streamlit application for Auto-BG.

Appendices

Calculating Nearest Neighbor Communities for Generation Evaluation

Auto-BG went through several iterations prior to 1.0 with multiple generation and evaluation strategies. One promising implementation that we believe holds potential future value for the application is evaluation through community aggregation. To calculate this we used a per-class Jaccard index[41]; given the sparsity of one-hot data for each game item, we needed a metric that could accurately measure the similarity of the input vector without being overwhelmed by 0/0 matches.

In this alternative class, after the input is parsed, an additional vector representation is returned. This vector passes to the Jaccard evaluation function which calculates the index within each of the four tag classes, checks the status of Cooperative, then finds the mean of these scores. We designed this per class check to mitigate the difference in magnitude between the available keys in each class and respect the relative value of sharing tags in a less diluted class such as Game Type. While our initial test implementation weighted all classes equally for the final score, we believe additional user feedback & testing would be required to find an ideal weighting scheme to optimize the output.

To actually find nearest neighbors, the above function is applied to a reduced source data set consisting of all game items that share at least three active tags or, if none exist, one active tag. To avoid incorrectly splitting tied scores, these translate to a ranking which is controlled by a top_n parameter in the class itself. With rankings applied, a new dataframe is returned that contains games up to the specified rank value ordered by closeness in the mean Jaccard index space to the input vector.

Auto-BG can, theoretically, use this information for several downstream evaluation tasks. We provide one example of this in the GitHub code; using Gensim's Document Similarity functions[42], we calculate the pairwise cosine similarity of the generated text to every community reference then weight those scores by related Jaccard index.

We found limited value in this as an absolute metric but it provides valuable insight on comparative evaluation between multiple texts generated from the same tag input parameters. One additional proposed use case included a real-time memorization check; if a prompt exactly matched a point of training data, we would expect that text to be included in the community. If a generated text begins over-performing when calculating document similarity, it could be discarded and regenerated, or logged for additional evaluation on root cause for overfitting training data.

We hope you find value in exploring and iterating on alternative versions of Auto-BG!

Metrics for Training GPT3 Model

We've provided detailed metrics from GPT3 training below. These values supplement the discussion on training methodology for text generation when working with the OpenAI API.

- 1298 steps
- 35309888 tokens (including repeats)
- 41536 elapsed examples (including repetitions)
- Median training loss score of 0.77 (IQR of around 0.245)
- Median training token accuracy of 0.464 (IQR of around 0.017)
- Throughout the entire training sequence, the sequence accuracy was 0, indicating that no predicted completions or full game board description sequences precisely matched the true completion tokens.[16]

GPT3 Model Architecture

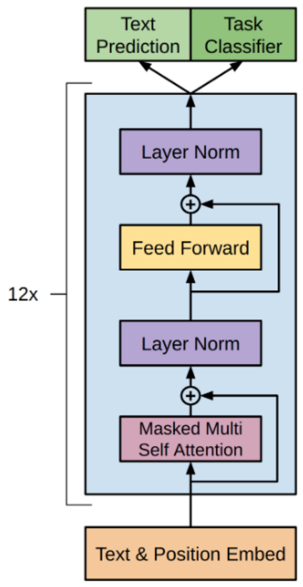


Figure 5: Decoder GPT Architecture[38]

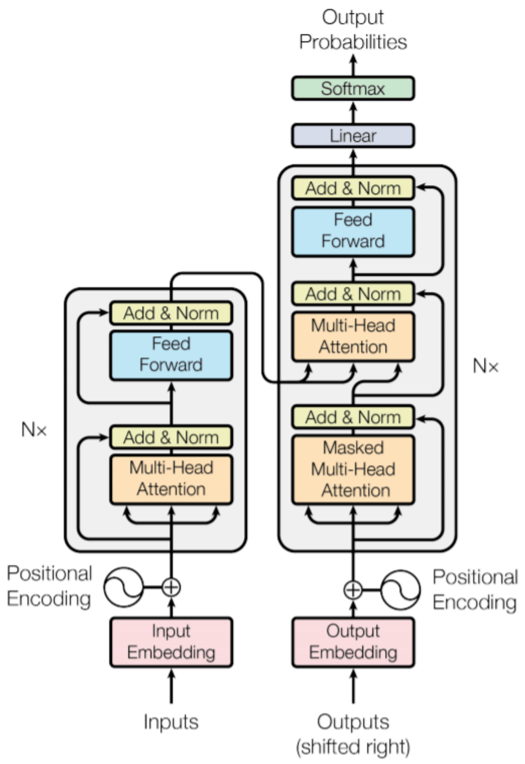


Figure 6: Encoder-Decoder Transformer Architecture[40]

A deep dive on the architecture of GPT3 is outside the scope of Auto-BG but for those who are unfamiliar and want a primer on how the model actually processes input prompts as a generator, we've provided this summary.

A transformer decoder structure (causal, or autoregressive, attention architectures) is the central component of GPT3, as the model does not include encoder attention blocks.[39] Only utilizing masked multi self attention, consisting of multiple layers of self-attention and feed-forward neural networks, allows the model to consider the context of the entire input sequence when making predictions.[38]

A brief overview of each stage:

1. Text & Position Embed: The first layer of a GPT-3 model tokenizes words using Byte Pair Encoding[43]. It converts these tokens into embeddings or represented vectors (initially sparse vectors into dense vectors)[38]. The model uses these to capture relative semantics of text and position of words within input sequences. Positional encoding is found by passing a token's relative position, represented by a scalar, through various sinusoidal functions at varying frequencies[43]. Overall, this stage

transforms input text into vector representations that the overall network can process. The embedding matrix maps each word in input text to a dense vector representation. This stage may also be viewed as a variation of positional and embedding stages in the decoder portion of the original transformer architecture[38].

2. Masked Multi-Head Self-Attention: This layer includes a self-attention step in which the decoder assigns weights, or ‘attention’ scores, to each state (the embeddings). Self-attention, in contrast to attention, allows the model to operate on all states in the same layer of the network[39]. Additionally, the model masks future text tokens to prevent itself from capturing future words or tokens during pre-training[39]. Also, multi-headed attention in general may apply independent transformations to each embedding to generate query, key, and value vectors[39]. This contributes to GPT-3’s ability to capture long-range dependencies between words in the input text[43].
3. Layer Normalization: Layer normalization is applied after each layer in the decoder. This post-layer normalization step normalizes the outputs of each layer by scaling and shifting them to have zero mean and unit variance[39], helping to prevent the vanishing or exploding gradient problem and improve convergence[39].
4. Feedforward Layer: Feedforward layers are fully connected neural networks applied in parallel to the output of the multi-head attention layer (after normalization)[39]. It transforms attention scores into a new representation more suitable for downstream tasks, such as text generation[43].
5. Output Embedding Layer: This final layer in the GPT-3 decoder transforms the output of repeated layer combinations, represented in figure 5, to a probability distribution over the vocabulary of possible output tokens[43]. This allows the decoder to generate coherent future text sequences[38].

References

1. “BoardGameGeek.” Accessed March 19, 2023. https://boardgamegeek.com/.
2. Canu, Nicholas, Kuan Chen, and Rhea Shetty. “Deconstructing Game Design.” Jupyter Notebook, October 20, 2022. https://github.com/canunj/deconstructing_games.
3. Danilák, Michal. “Langdetect.” Python, March 15, 2023. https://github.com/Mimino666/langdetect.
4. GitLab. “Recommend.Games / Board Game Scraper · GitLab,” March 10, 2023. https://gitlab.com/recommend.games/board-game-scraper.
5. Canu, Nicholas, Jonathan Ellis, and Oldenkamp, Adam. “The Impact of Crowdfunding on Board Games.” Jupyter Notebook, May 20, 2022. https://github.com/canunj/BGG_KS_Analysis.
6. Spielberg, Steven dir. *Jurassic Park*. 1993; Universal City, CA: Universal Studios, 2022. UHD Blu Ray.
7. Chiu, Ke-Li, Annie Collins, and Rohan Alexander. “Detecting Hate Speech with GPT-3.” arXiv, March 24, 2022. https://doi.org/10.48550/arXiv.2103.12407.
8. Lucy, Li, and David Bamman. “Gender and Representation Bias in GPT-3 Generated Stories.” In *Proceedings of the Third Workshop on Narrative Understanding*, 48–55. Virtual: Association for Computational Linguistics, 2021. https://doi.org/10.18653/v1/2021.nuse-1.5.
9. Abid, Abubakar, Maheen Farooqi, and James Zou. “Persistent Anti-Muslim Bias in Large Language Models.” In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, 298–306. AIES ’21. New York, NY, USA: Association for Computing Machinery, 2021. https://doi.org/10.1145/3461702.3462624.
10. Schramowski, Patrick, Cigdem Turan, Nico Andersen, Constantin A. Rothkopf, and Kristian Kersting. “Large Pre-Trained Language Models Contain Human-like Biases of What Is Right and Wrong to Do.” *Nature Machine Intelligence* 4, no. 3 (March 2022): 258–68. https://doi.org/10.1038/s42256-022-00458-8.
11. Huang, Po-Sen, Huan Zhang, Ray Jiang, Robert Stanforth, Johannes Welbl, Jack Rae, Vishal Maini, Dani Yogatama, and Pushmeet Kohli. “Reducing Sentiment Bias in Language Models via Counterfactual Evaluation.” arXiv, October 8, 2020. https://doi.org/10.48550/arXiv.1911.03064.
12. Cao, Yang Trista, and Hal Daumé III. “Toward Gender-Inclusive Coreference Resolution.” In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 4568–95, 2020. https://doi.org/10.18653/v1/2020.acl-main.418.
13. Floridi, Luciano, and Massimo Chiriatti. “GPT-3: Its Nature, Scope, Limits, and Consequences.” *Minds and Machines* 30, no. 4 (December 1, 2020): 681–94. https://doi.org/10.1007/s11023-020-09548-1.
14. Thorp, H. Holden. “ChatGPT Is Fun, but Not an Author.” *Science* 379, no. 6630 (January 27, 2023): 313–313. https://doi.org/10.1126/science.adg7879.
15. Zinkula, Aaron Mok, Jacob. “ChatGPT May Be Coming for Our Jobs. Here Are the 10 Roles That AI Is Most Likely to Replace.” Business Insider. Accessed March 19, 2023.

https://www.businessinsider.com/chatgpt-jobs-at-risk-replacement-artificial-intelligence-ai-labor-trends-2023-02.

16. "Fine-Tuning." Accessed March 23, 2023. https://platform.openai.com/docs/guides/fine-tuning/conditional-generation.
17. "Best Practices for Prompt Engineering with OpenAI API | OpenAI Help Center." Accessed March 26, 2023. https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-openai-api.
18. Gunawan, D., C. A. Sembiring, and M. A. Budiman. "The Implementation of Cosine Similarity to Calculate Text Relevance between Two Documents." *Journal of Physics: Conference Series* 978, no. 1 (March 2018): 012120. https://doi.org/10.1088/1742-6596/978/1/012120.
19. Token. "Token · SpaCy API Documentation." Accessed March 26, 2023. https://spacy.io/api/token#similarity.
20. Reynolds, Laria, and Kyle McDonell. "Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm." arXiv, February 15, 2021. https://doi.org/10.48550/arXiv.2102.07350.
21. "OpenAI API Reference." Accessed March 23, 2023. https://platform.openai.com/docs/api-reference/completions/create.
22. "BERT Score - a Hugging Face Space by Evaluate-Metric." Accessed April 4, 2023. https://huggingface.co/spaces/evaluate-metric/bertscore.
23. "ROUGE - a Hugging Face Space by Evaluate-Metric." Accessed April 4, 2023. https://huggingface.co/spaces/evaluate-metric/rouge.
24. Sellam, Thibault, Dipanjan Das, and Ankur P. Parikh. "BLEURT: Learning Robust Metrics for Text Generation." arXiv, May 21, 2020. https://doi.org/10.48550/arXiv.2004.04696.
25. Zhou, Yongchao, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. "Large Language Models Are Human-Level Prompt Engineers." arXiv, March 10, 2023. https://doi.org/10.48550/arXiv.2211.01910.
26. MasterClass. "How to Write a Book Title in 7 Tips: Create the Best Book Title - 2023." Accessed March 28, 2023. https://www.masterclass.com/articles/how-to-write-a-book-title-in-7-tips-create-the-best-book-title.
27. Qualtrics. "How to Find the Perfect Product Name in 2023." Accessed March 28, 2023. https://www.qualtrics.com/experience-management/product/product-naming/.
28. Raffel, Colin, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer." arXiv, July 28, 2020. http://arxiv.org/abs/1910.10683.
29. "Michau/T5-Base-En-Generate-Headline · Hugging Face." Accessed March 28, 2023. https://huggingface.co/Michau/t5-base-en-generate-headline.
30. "T5-Base · Hugging Face," November 3, 2022. https://huggingface.co/t5-base.

31. Tay, Yi, Mostafa Dehghani, Jinfeng Rao, William Fedus, Samira Abnar, Hyung Won Chung, Sharan Narang, Dani Yogatama, Ashish Vaswani, and Donald Metzler. "Scale Efficiently: Insights from Pre-Training and Fine-Tuning Transformers." arXiv, January 30, 2022. https://doi.org/10.48550/arXiv.2109.10686.
32. Kusner, Matt J., Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. "From Word Embeddings to Document Distances." In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, 957–66. ICML'15. Lille, France: JMLR.org, 2015.
33. Google Trends. "Google Trends." Accessed April 4, 2023. https://trends.google.com/trends/explore?geo=US&q=prompt%20engineering&hl=en.
34. spielen.de. "Brettspiele & Gesellschaftsspiele." Accessed April 2, 2023. https://gesellschaftsspiele.spielen.de/.
35. Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, Dario Amodei. "Language Models are Few-Shot Learners." arXiv, May 28, 2020. Accessed March 12, 2023. https://arxiv.org/abs/2005.14165
36. "About." Accessed April 1, 2023. https://openai.com/about
37. Alec Radford, Jeffrey Wu, Rewon Child, David Lua, Dario Amodei, Ilya Sutskever. "Language Models are Unsupervised Multitask Learners." Accessed March 2, 2023. https://openai.com/research/better-language-models
38. Alec Radford, Karthik Narshimhan, Tim Salimans, Ilya Sutskever. "Improving Language Understanding by Generative Pre-Training." Accessed March 2, 2023. https://openai.com/research/language-unsupervised
39. Lewis Tunstall, Leandro von Werra, Thomas Wolf. *Natural Language Processing with Transformers: Building Language Applications with Hugging Face.* 2022.
40. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. "Attention is All You Need." arXiv, Jun 12, 2017. Accessed March 2, 2023. https://arxiv.org/abs/1706.03762
41. scikit-learn. "Sklearn.Metrics.Jaccard_score." Accessed April 12, 2023. https://scikit-learn/stable/modules/generated/sklearn.metrics.jaccard_score.html.
42. "Gensim: Topic Modelling for Humans." Accessed April 12, 2023. https://radimrehurek.com/gensim/similarities/docsim.html.
43. "The GPT-3 Architecture, on a Napkin." Accessed April 8, 2023. https://dugas.ch/artificial_curiosity/GPT_architecture.html