Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

Sample-Efficient Behavior Cloning Using General Domain Knowledge

Feiyu Zhu, **Jean Oh** and **Reid Simmons**
Carnegie Mellon University
{feiyuz, rsimmons}@andrew.cmu.edu, jeanoh@cmu.edu

Abstract

Behavior cloning has shown success in many sequential decision-making tasks by learning from expert demonstrations, yet they can be very sample inefficient and fail to generalize to unseen scenarios. One approach to these problems is to introduce general domain knowledge, such that the policy can focus on the essential features and may generalize to unseen states by applying that knowledge. Although this knowledge is easy to acquire from the experts, it is hard to be combined with learning from individual examples due to the lack of semantic structure in neural networks and the time-consuming nature of feature engineering. To enable learning from both general knowledge and specific demonstration trajectories, we use a large language model's coding capability to instantiate a policy structure based on expert domain knowledge expressed in natural language and tune the parameters in the policy with demonstrations. We name this approach the Knowledge Informed Model (KIM) as the structure reflects the semantics of expert knowledge. In our experiments with lunar lander and car racing tasks, our approach learns to solve the tasks with as few as 5 demonstrations and is robust to action noise, outperforming the baseline model without domain knowledge. This indicates that with the help of large language models, we can incorporate domain knowledge into the structure of the policy, increasing sample efficiency for behavior cloning.

1 Introduction

Behavior cloning and its variants have demonstrated success in learning policies for autonomous driving [Hu *et al.*, 2022], table-top manipulation [Chi *et al.*, 2023], household tasks [Fu *et al.*, 2024], and so on. Yet, due to a distribution mismatch between expert trajectories and the states encountered during deployment [Osa *et al.*, 2018] as well as the increase in model size, they often rely on a large number of expert demonstrations to learn a robust policy [Zhao *et al.*, 2024] and cannot generalize well to new camera poses, unseen distractor objects, novel background texture etc. [Xie *et al.*, 2024].

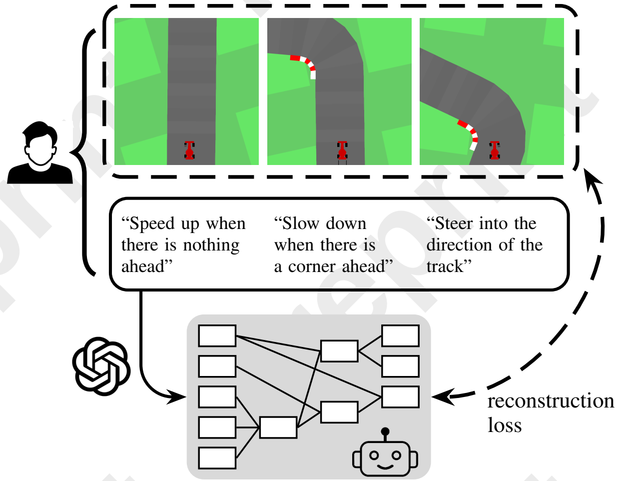

Figure 1: Overview of behavior cloning with general domain knowledge. We collect domain knowledge from the expert (middle) in addition to demonstrations (top). An LLM translates this knowledge into the structure of the policy (bottom) and behavior cloning is used to learn the parameters of the policy from the demonstrations.

Despite the vast potential variations of a task, the underlying principles of solving it often stay the same. For example, when trying to open a door, the motion depends only on the position, type of the handle, and the direction the door is expected to open. Furthermore, the opening direction of the door can be inferred from the location of the hinges. This general domain knowledge naturally reflects the latent features of the task and their connectivities: *direction* is a latent variable that depends on the location of the hinge, and the specific motion depends on *direction* but not other features such as color or size. It has been shown that following general knowledge enables zero-shot transfer to novel environments for tasks in discrete domains [Zhu and Simmons, 2024].

Although it is relatively easy for a domain expert to explain the general ideas, it is challenging for them to specify the detailed instructions, especially for continuous action spaces. Additionally, unstructured model architectures have very few semantic structures, creating a barrier between domain knowledge expressed in natural languages and the internal representation of a learning model. Therefore, existing work that attempts to integrate domain knowledge focuses mainly on state abstractions that highlight the important features of the task [Peng *et al.*, 2024].

To make use of domain knowledge beyond state representations, we propose ***Knowledge Informed Models (KIM)*** (Figure 1) to take advantage of the coding capabilities of LLMs to instantiate the entire *structure* of the policy, while using expert demonstrations to fit the unspecified parameters in the policy (e.g., how much to slow down when approaching the corner). This allows the model to tailor not only which input features are used, but also how latent variables should be defined and computed. This semantically meaningful structure has fewer parameters to be tuned and guides the policy to interpret the demonstrations strategically, so they inherently require fewer samples and are less susceptible to overfitting.

The contributions of this paper are twofold: 1) we propose an approach to make use of general domain knowledge to enable sample-efficient behavior cloning, and 2) we demonstrate the effectiveness and robustness of our approach in continuous environments with discrete and continuous action spaces with very few demonstrations. Specifically, our approach achieves better performance than the unstructured baseline with statistical significance and degrades much less than the baseline under the noisy action condition. [1]

2 Related Work

2.1 Sample-Efficient Behavior Cloning

Data augmentation is a common technique for expanding the coverage of expert demonstrations [Ankile *et al.*, 2024], often using visual synthesis [Zhou *et al.*, 2023], local continuity [Deshpande *et al.*, 2024], time-reversal symmetry [Cheng *et al.*, 2024]. Extending this direction, other works proposed to learn a local model to guide the policy from unseen states to known states [Park and Wong, 2022] or to learn a world model [Kolev *et al.*, 2024]. Another similar approach is to use state abstraction to hide the irrelevant features of the states [Peng *et al.*, 2024] such that the policy will not be conditioned on them without the need for data augmentation. Other approaches include using a better representation of actions [Chi *et al.*, 2023], building up skill libraries to reuse previously learned skills [Wan *et al.*, 2024], instructing the expert to demonstrate failure recovery [Brandfonbrener *et al.*, 2023].

Unlike previous work that mainly focused on sample-level operations, our work is the most similar to [Mao *et al.*, 2023] where we aim to improve sample efficiency by specializing the structure of the policy being learned to the specific task and its relevant features. However, instead of searching through a pre-defined architecture space or merely abstracting the state representations, we take advantage of experts' domain knowledge to instantiate a neural net with a specific structure that is specialized to the task as the policy model.

2.2 LLM Assisted Policy Learning

Previous work has used the coding capability of LLMs to implement agent policies [Zhu and Simmons, 2024], represent world models [Tang *et al.*, 2024b], generate reward distributions [Bucker *et al.*, 2024], and translate underspecified task specifications into structured representations [Liu *et al.*, 2023]. However, the codes generated are mostly symbolic, relying on well-defined APIs to execute the policy. They are also static, allowing very little post-generation adaptation. Others have used LLMs to generate target action distribution [Zhou *et al.*, 2024] or provide reward signals [Wang *et al.*, 2024] that can be used to train smaller models. But unlike the coding-focused approaches that can make use of external knowledge, these sample-based methods depend solely on the pre-trained knowledge in the LLMs.

Our work makes use of the coding ability of LLMs but also enables parameter tuning after code generation. This alleviates the dependence on the LLMs to get everything correct in one go and makes it possible to incorporate expert knowledge that is not captured by the LLMs.

2.3 Knowledge Integration in Machine Learning

It is well-acknowledged that integrating existing human knowledge helps with machine learning [Deng *et al.*, 2020], where human knowledge is commonly in the form of feature selections and invariance in the task.

Prior to the popularity of learning feature representations, models were trained with features that were picked manually [Bahnsen *et al.*, 2016] or according to some statistical metrics [Ghojogh *et al.*, 2019]. Despite achieving great performance in complex tasks such as planning for driving [Dauner *et al.*, 2023], existing works modify only the inputs to the models but not the architecture of the models, not taking full advantage of the existing domain knowledge.

Other works have developed specialized architecture that incorporates the invariance in the task, including SE(3)-equivariant layers for tabletop manipulation [Eisner *et al.*, 2024] and drug discovery [Schneuing *et al.*, 2024], and physics-informed neural nets that respect PDE constraints [Wang *et al.*, 2023]. These approaches require the experts to have both domain knowledge for the task and also engineering skills for model development, and the architecture can only be used in a certain family of tasks.

By contrast, our approach takes advantage of LLMs' coding skills to implement arbitrary general domain knowledge expressed in natural languages, making it more accessible to make use of existing human knowledge. And it implements the architecture from the ground up, reflecting both the selection of features and the connections between the features.

3 Knowledge Informed Model (KIM)

3.1 Structured Policy

In this work, we use the term "structured policy" to refer to a model in which latent variables and their connectivities are specialized to the task. The latent variables typically have semantic meanings, representing key features of the task that are not directly accessible from the input. A structured policy has many distinctions compared to an unstructured model such as a generic multi-layer perception (MLP). It takes advantage of the sparsity that exists in many domains [Mao *et al.*, 2023] and assigns learnable parameters to the related latent variables instead of all pairs of latent variables. It may also contain a variety of operations (e.g., `max`, `clip`) that are

[1] Code at github.com/zfy0314/knowledge-informed-model

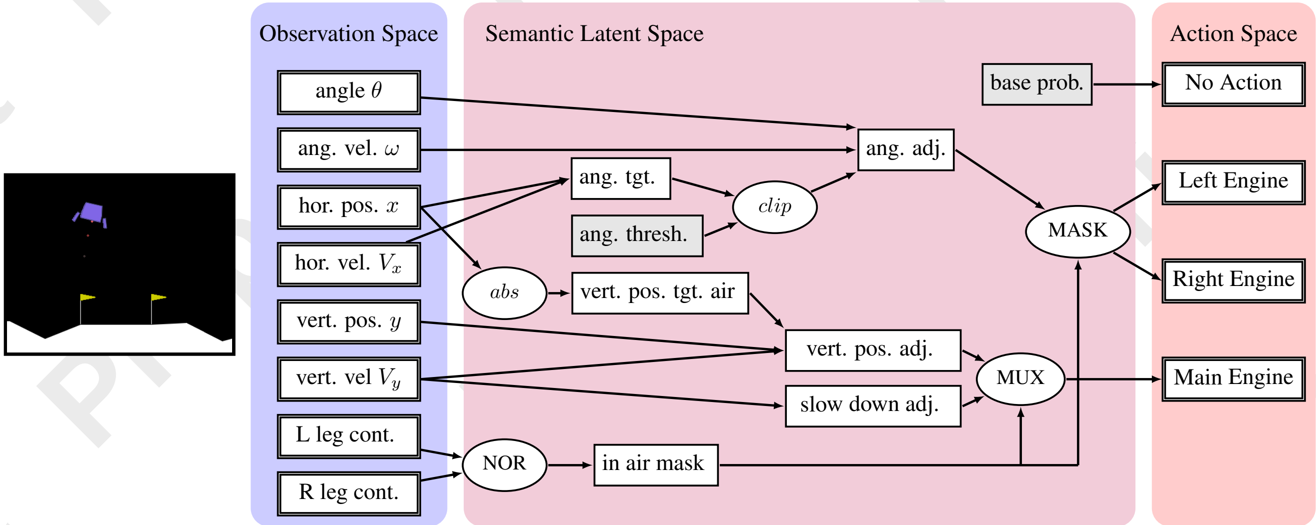

Figure 2: Illustration of the Knowledge Informed Model for the Lunar Lander environment generated by GPT. Each box represents a variable, with the white boxes representing latent variables and the gray boxes representing tunable parameters in the model. Arrows represent the dependencies between variables and each has an associated learnable weight. The oval shapes represent non-linear operations. By default the value of latent variables is a linear combination of the variables that it depends on.

beyond linear transformations and primitive non-linear activations. As a result, the policy structure is a highly concise representation of the general structure of a solution to the task (e.g., Figure 2). Mathematically, a policy structure can be represented as a 4-tuple $\langle V, O, E, \Theta\rangle$ where

- V is a set of nodes that each represent a latent variable.
- O is a set of nodes that each represent an instance of an operation (e.g., the `clip` function).
- $E : \{\langle u_i, v_i\rangle\}$ is a set of edges that represents the dependencies between latent variables and operations.
- $\Theta : E \to \mathbb{R}$ is a set of weights associated for each edge.

In general, it can be seen as a weighted acyclic graph.

During inference time, latent variables are computed in an order based on the partial order of their dependencies. That is, the latent variables that only depend on the input features are computed first (1st degree latent), then the variables that only depend on the input features and the 1st degree latent, and so on. The specific values of the variables depend on the operations defined by the policy structure and the weight parameters connecting the latent variables. For instance, as shown in Figure 2, the value of angle adjustment is a linear combination (with bias) of the current angle, current angular velocity, and the angle target after clipping.

The weights in Θ can be updated via gradient descent by supervised learning on the action output, similar to how a typical MLP can be learned. This enables numeric learning in structured policies. Section 3.3 provides a detailed explanation of the learning process.

Although a structured policy along with its parameter values can be directly coded by an expert (e.g., the heuristics-based policy in the Lunar Lander task [Towers *et al.*, 2024]), doing so manually is typically time-consuming. Therefore, to enable more scalability, it would be beneficial that the structure of the policy be generated from natural language descriptions and the parameters be learned from a few demonstrations, which are easier to acquire from an expert.

3.2 KIM Generation via LLM

We assume access to domain knowledge $\mathcal{K}$ from an expert in natural language that describes the high-level ideas that guide the demonstrations $\mathcal{D}$. This is attainable as previous works have shown that humans typically construct simplified mental representations during problem-solving [Ho *et al.*, 2022], so they should also be able to articulate the general knowledge used to perform the demonstrations.

In practice, we collect the general description of the strategy used by the expert, the feature space and the action space of the task, and any additional information about the environment that might be useful. We can instantiate a policy structure based on the provided general knowledge using an LLM (e.g., GPT4o [OpenAI, 2024]). That is

$$\langle V, O, E, \Theta_{\text{init}}\rangle = LLM(\mathcal{S} + \mathcal{K}) \tag{1}$$

where $\mathcal{S}$ is the system prompt that is shared for all tasks.

Concretely, we use the chain-of-thought prompting [Wei *et al.*, 2023] to instruct the LLM to implement the models. First, it is instructed to extract all the input features and latent variables in the general knowledge description and list their type and shape (e.g., angle target has type `float` and shape `(1,)`).

Next, the LLM is expected to re-arrange the latent variables in the order in which they should be computed based on variable dependencies (e.g., `angle target` should appear before `angle adjustment` as the latter depends on the former). And, for each of them, list all the previously computed variables that the current variable depends on and what operators are needed to connect them. Empirical experiments showed that without this step the LLM may miss some of the connections in the code generation process. During this process, the LLM is also instructed to classify each connection between latent variables as "positively correlated" or "negatively correlated" based on the expert knowledge (e.g., "the target angle depends on the horizontal position and should point to the center" indicates that the target angle and the current horizontal position are positively correlated). This information can be used to set the initial value of the parameters Θ_{init}. Since the general knowledge does not contain specific numeric relationships, we let the LLM set very rough initial values (e.g., -0.1 for negatively correlated variables). As the model structure reflects the semantic meaning of the expert's strategy, it does not have the permutation symmetry as many unstructured models do [Ainsworth *et al.*, 2022] and hence is more sensitive to the initial values.

The final step is to implement the structure as a subclass of `nn.Module` in PyTorch. When coding the model, the LLM is instructed to classify all the parameters as gradient or non-gradient, where the non-gradient parameters are those that cannot be learned using gradient descent (e.g., the bounds of a `clip` function) whereas the rest are gradient parameters.

Figure 2 shows an example of a model generated for the Lunar Lander task. The prompts that were used to generate this model can be found in the Appendix. Note that we do not include any examples in the prompt, and fully leverage the zero-shot coding capability of the LLM.

3.3 Behavior Cloning for KIM

After the model structure is set, we train the parameters using the standard behavior cloning objective to tune the parameters Θ in the policy.

$$\min_{\Theta} \mathbb{E}_{\langle s_i, a_i \rangle \in \mathcal{D}} \mathcal{L}(a_i, \pi_\Theta(s_i)) \tag{2}$$

In practice, we use grid search over the values for the non-gradient parameters. For each combination of the non-gradient parameter values, we use gradient descent to optimize the remaining gradient parameters. By default, we use cross-entropy loss for discrete action spaces and mean square error for continuous action spaces. The combination (both non-gradient and gradient) that achieves the least overall loss is kept as the final model parameter.

Because the connections between latent variables are sparse, the number of total parameters is small compared to unstructured models. Additionally, we focus on using only a few demonstrations. Therefore, we can perform gradient descent on all the demonstrations at once for most tasks without having to separate the samples into mini-batches. This helps to stabilize the training process.

Unlike unstructured models, latent variables in KIM have semantic meanings as they are extracted from the provided expert knowledge. Therefore, it is possible for the expert to directly set the value of some constant parameters, or the weights connecting different latent variables. This will make learning easier since there are fewer parameters to optimize.

4 Experiments

We experiment with the Lunar Lander and Car Racing environments in Gymnasium [Towers *et al.*, 2024], covering both discrete and continuous action spaces. We used `gpt-4o-2024-11-20` as our LLM for the experiments.

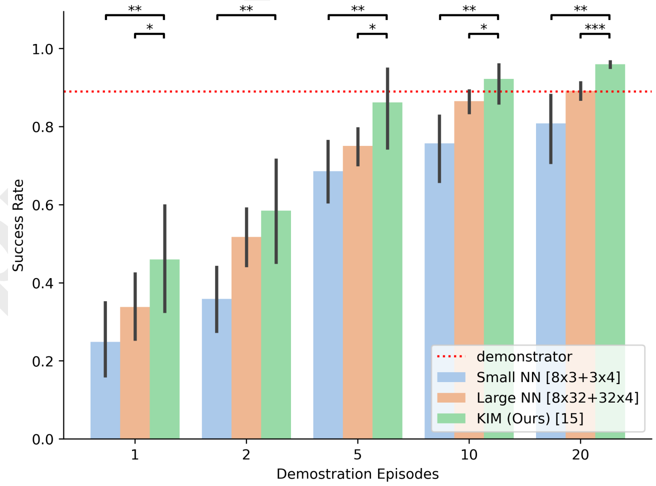

Figure 3: Success rates in the Lunar Lander task, evaluated on 100 random start states per session. The error bars in the plot show the 95% confidence interval estimated by 20 sets of demonstration episodes. Asterisks denote the statistical significance levels of paired t-tests (* for < 0.05, ** for < 0.01, and *** for < 0.001).

4.1 Lunar Lander

The objective of the Lunar Lander task is to control the engines of the lander to perform a soft landing on the landing pad (an illustration can be found in Figure 2). The observation space is the horizontal position and velocity, vertical position and velocity, angular position and velocity, and whether each of the landing legs is in contact with the surface. The last two features on the landing legs are binary, while the others are continuous. Each new episode has a different randomly initialized starting configuration. The action space is discrete, consisting of doing nothing or activating one of the left, main, or right engines. The episode ends if the lander lands safely, crashes, or runs out of fuel after 1000 steps. Typically, a successful landing can be achieved in around 200 steps.

We use the heuristic policy defined in the Gymnasium package as the expert policy that generates demonstrations. This policy achieves around 89% success rate in the environment, however, we keep only the successful episodes as demonstrations for training. We manually describe the strategy of the heuristic policy as the expert general knowledge and use it to prompt the LLM for KIM generation. The prompt can be found in the Appendix.

For the baseline condition, we use an MLP and formulate it as a classification problem with cross-entropy loss, where the objective is to predict which action the expert policy is going to perform given all of the features of a state. For both conditions, we randomly sample 20% of the demonstration steps as the validation set, and keep the model parameters with the least loss in the validation set for evaluation.

4.2 Car Racing

The objective of the car racing task is to complete a winding track as fast as possible (an illustration is provided in Figure 1). The track is defined by a sequence of tiles that span from the left of the track to the right. The reward is defined as

$$1000 * N - 0.1 * T \tag{3}$$

where N is the percentage of the tiles on the track covered within 1000 steps and the T is the number of time steps taken to complete the track (or truncated at 1000 if the race car runs out of time). A tile is covered if at least one of the wheels makes contact with it. In addition to the first 1000 steps, after recording the reward for the environment, we keep running the environment for another 2000 steps to collect the maximum coverage of the track for a policy. Empirically this is sufficient to wait for the policy to finish the track at least once.

The original environment features an image-based observation space. To bypass the perception challenges, we use a basic representation where each state is defined by a sequence of tiles (their mid-point coordinates, angular heading, the difference in coordinates and headings compared to the previous tile, whether the tile has a corner marker) that makes up the visible tracks in the current frame and the current state of the race car (including current speed, direction or steer, value of the gyroscope, and ABS sensors on each of the wheels). Each new episode features a new track layout. The action space is continuous and consists of the steering of the race car, the engagement of the gas pedal, and the engagement of the brake pedal. The gas pedal only acts on the rear wheels while the brakes are on all wheels. The exact dynamics of the race car are unavailable to the human expert and the model.

In this environment, the domain knowledge and demonstration trajectories all come from a human researcher who has interacted with the environment extensively. Specifically, the actions are collected using a Logitech controller for continuous actions while the researcher is looking at the rendering shown on a screen. A total of 200 demonstration episodes are collected through multiple sessions. This setup reflects the real-world scenario where the same expert provides the domain knowledge while giving demonstrations that correspond to the domain knowledge. Since there are human errors during execution, inconsistencies between different sessions, and discrepancy in the perception modalities (i.e., the policy takes in low dimensional input while the human expert operates on images), there will be noise in the demonstration provided, further resembling real-world settings. All demonstrations achieve perfect coverage (i.e., the race car never goes off the track), and have an average reward of 913.5.

The baseline condition is an MLP that takes the same basic representation as input. Its learning objective is to minimize the mean squared error between its output and the expert action in the demonstration set. Similar to the previous environment, 20% of the demonstrations are reserved for validation while the rest are used for training in both conditions.

4.3 Results on Learning with a Few Demonstrations

Figure 3 shows the success rate in the Lunar Lander task between using KIM and the baseline neural networks of two different sizes (the number of parameters are listed in the legend). The figure shows that given a fixed number of demonstrations, KIM, with only 15 parameters, achieves a $20\%+$ higher success rate than the small NN that has a similar number of parameters and a $7\%+$ higher success rate than the larger NN that has 25x more parameters. This shows that given the same demonstrations, KIM learns a more robust policy. All but one pair of comparisons show statistical significance in a paired t-test where the pairing is based on having the same set of demonstrations.

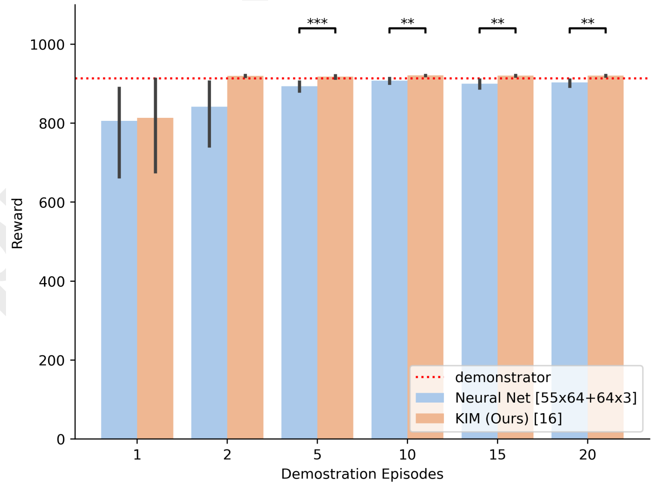

Figure 4: Reward in the Car Racing task, evaluated on 100 random tracks per session. The error bars in the plot show the 95% confidence interval estimated by 10 sets of demonstration episodes.

Figure 4 shows the reward comparisons between KIM and a neural net in the Car Racing task. It shows that starting from as few as 2 demonstrations KIM yields good performance and low variance. This attributes to KIM having very few parameters organized in a semantically meaningful structure and is thus more robust to imperfect demonstrations. When there are more demonstrations, KIM still outperforms the baseline (which has 200x more parameters) with statistical significance. Overall, the plots show that despite the demonstrations and the provided general knowledge may not be perfectly aligned, using the knowledge to instantiate the model still leads to better performance.

4.4 Result on Environments with Noise

To evaluate how well KIM does in noisy settings, we randomly corrupt its output with a Gaussian noise when it is interacting with the environment. That is,

$$\mathbf{a}_{\text{new}} \sim \mathcal{N}(\mathbf{a}_{\text{pred}};\ \text{noise\_level} \cdot \mathbf{I}) \tag{4}$$

Note that in this setting the models are still trained with expert demonstrations captured in a noise-free environment. The noise is only added after the model is trained.

Figure 5 shows the comparison between KIM and the baseline (using the 10 models trained on 10 demonstrations each in the previous section) in environments with different noise levels. It shows that as the noise level increases, the performance of the baseline condition degrades much more drastically than KIM. In particular, KIM can still retain around 65% of the reward even with considerable noise (the action space is $[-1, 1]$ and the Gaussian noise has a standard devi-

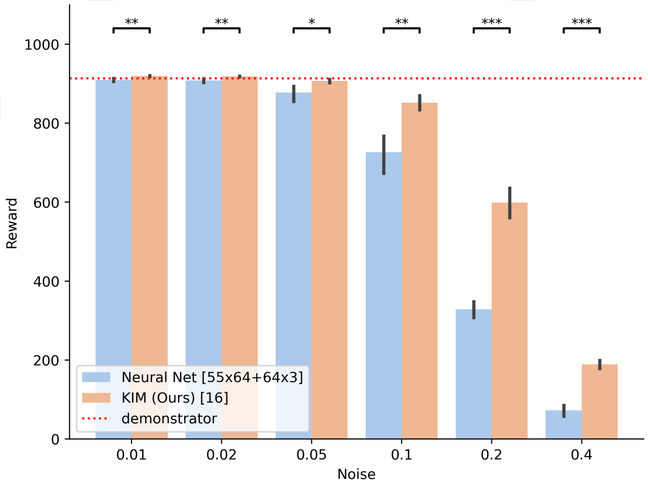

Figure 5: Reward in the Car Racing task with different levels of action noise, each evaluated on 100 random tracks per session. The error bars in the plot show the 95% confidence interval estimated by 10 sets of demonstration episodes. Each model is trained with 10 demonstration episodes.

Listing 1: Code snippet generated by GPT on steering control

```
steer_control = (
    self.steer_weight *
    (target_heading - current_heading) *
    (1 - current_speed)
)
```

ation of 0.2). This illustrates that optimizing with respect to general domain knowledge makes the policy less brittle.

4.5 Qualitative Analysis

The high variance in the neural net baseline learned with 2 demonstrations is partially caused by the learned model losing control of the race car and swirling off the track.

Figure 6 shows an example where the baseline condition loses control while KIM steers the race car to stay on the track. This is because the knowledge of “don’t steer too drastically when accelerating” is crucial for driving the rear-wheel drive race car in this domain, yet is only implicitly illustrated through the expert demonstrations. As a result, when the demonstrations are not sufficiently indicative of this, an unstructured model may miss this constraint, leading to catastrophic outcomes, especially if no expert demonstration illustrates how to regain control after losing traction.

However, KIM provides another channel (i.e., the general knowledge in natural language) for the expert to pass knowledge to the learning model. Furthermore, these domain constraints are enforced by the structure of the model such that it is more robust to imperfect demonstrations. Therefore it can navigate the corner much more smoothly. Listing 1 shows the structure that is informed by the following instructions:

> the output of steering should be scaled based on the current speed such that when speed approaches 1 the steer magnitude should approach 0.

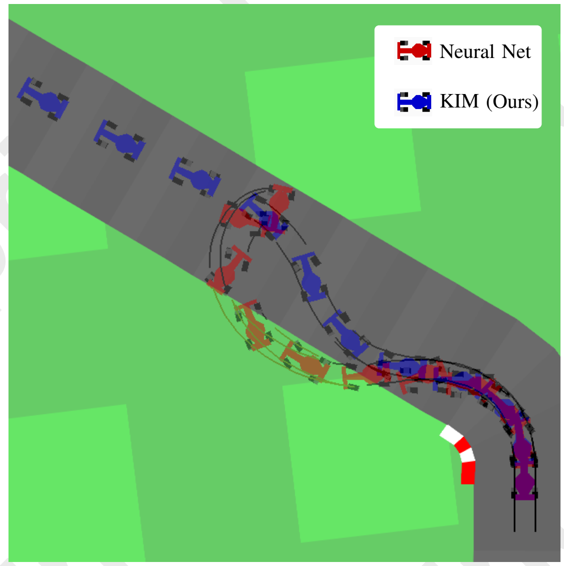

Figure 6: Qualitative samples from the neural net baseline (red) and KIM (blue) in the same starting condition. The two models are trained on the same set of 2 demonstrations. The positions of the race car are captured with a fixed time interval in between. The baseline over-steers and loses control while KIM corrects the heading of the race car.

The full prompt and model can be found in the Appendix.

We observe a similar case where the race car driven by KIM follows the center line more closely, following the expert’s instructions, while the baseline neural network takes the corners more tightly (Figure 7). This helps explain why KIM is more robust to action noise, since it is less likely to leave the track.

4.6 Additional Comparisons

Table 1 shows the ablation on different settings of KIM in the two environments.

The human-generated code condition is where the code generated by GPT is replaced with code generated by a human researcher given the same prompt. Overall, the codes generated are very similar and hence the performance is similar in both tasks. The differences in the code implementation are very subtle. For example, the human-generated code set `bias=False` for the linear layer for adjusting for steering because by default the race car should go straight. However, GPT did not make use of this information and defined the linear layer with bias. Details like this likely lead to a slightly smaller variance in the human-generated code condition.

On the other hand, there is a distinctive difference between having pre-filled initialization values for the parameters and not. In the random initialization condition, all parameters are sampled from a standard normal distribution. The result shows very high variance, confirming that the optimization space for KIM does not have the “one basin” phenomenon [Ainsworth *et al.*, 2022] that helps optimize typical unstructured models. So prompting the LLM to analyze the relationship between latent variables is vital for performance.

Additionally, KIM performed slightly better than the ex-

	Lunar Lander Success Rate ↑	Car Racing Reward ↑	Car Racing Coverage ↑
KIM	0.891 (±0.184)	921.631 (±10.251)	1.000 (±0.002)
KIM w/ human-generated code	0.880 (±0.086)	922.048 (± 7.892)	1.0 (±0.0)
KIM w/ random parameters	0.702 (±0.390)	824.371 (±79.916)	0.999 (±0.005)
Expert policy	0.890	913.498 (±10.86)	1.0 (±0.0)

Table 1: Comparison of KIM in different settings. KIMs are trained with 10 demonstrations. For the Lunar Lander environment, the success rate is computed among 100 randomly initialized configurations. We learn 10 models each trained with a different set of demonstrations to estimate the mean and standard deviation of success rates. For the Car Racing environment, the mean and standard deviation are evaluated on 100 randomly initialized track layouts for a single model.

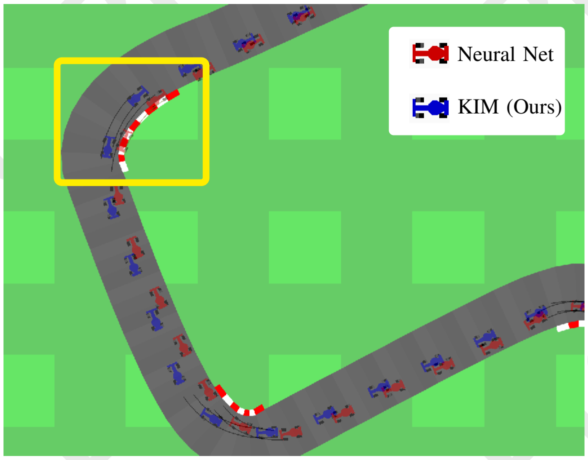

Figure 7: Stability comparison between the two conditions. Both models are trained on the same set of 10 demonstrations. The baseline takes the corner very tightly (it is on the curb in the yellow box) while KIM is closer to the center line. Despite traveling for a longer distance, KIM takes less time to navigate through the two corners.

pert demonstration in the Car Racing environment. This is likely due to the imperfections in the human demonstrations and KIM's ability to filter those imperfections in the demonstrations based on general knowledge, which is hard for an unstructured model as it treats all demonstrations equally.

5 Discussion

5.1 Limitations

The current method relies heavily on having access to good expert instructions. These are relatively easy to acquire from well-established settings (e.g., assembly lines or aircraft controls) but could be hard in scenarios that require more nuance (e.g., social navigation). It is also challenging for human experts to provide exhaustive instructions in one go, or if the action space is different between human experts and the policy (e.g., learning a quadruped robot walking policy). Also, current work searches through non-gradient parameters which can be costly. This can be partly resolved by using parameter tuning techniques or specifying the range for the searches.

Another limitation is the dependency on LLMs' zero-shot coding capabilities. In addition to the potential misalignment issue [Greenblatt *et al.*, 2024], all contemporary LLMs operate on input sequences with a length limit, making it impossible to generate a KIM if the general domain knowledge exceeds that limit. Additionally, previous work has reported that LLMs may neglect information in a long prompt [Liu *et al.*, 2024], which may lead to a suboptimal structure.

5.2 Future Work

Since it is hard to specify all the general knowledge all at once, an extension is to support interactive and incremental KIM. This would require integrating the code editing capability of LLM [Tang *et al.*, 2024a] and modifying the existing structure based on the incoming knowledge. This also helps if the LLM does not generate the correct code the first time, by giving the expert opportunities to amend the generated model.

To tackle more complex tasks, a promising direction is to first use the proposed approach to learn a library of action primitives, then learn to complete more complex tasks by referencing and reusing those action primitives. As each action primitive only requires a few demonstrations, and LLMs can perform general task decomposition [Zhu and Simmons, 2024], we can learn complex skills

Additionally, the benefit of integrating general knowledge applies beyond representing policies - it can also be used to represent the transition model of the world. One could use a similar technique to develop a sample-efficient model-based reinforcement learning policy where the structure of the world comes from existing database or humans and the specific parameters are tuned by interacting with the real world.

6 Conclusion

In this work, we proposed Knowledge Informed Models (KIM) that combine expert demonstrations with general domain knowledge by instantiating a policy structure from the general knowledge before tuning its parameters with expert demonstrations. This bridges the gap between the semantic knowledge human experts typically possess and the unstructured model architectures that are used for behavior cloning. We detailed how an LLM can be used to enable structure generation and how it could be learned from gradient descent once the initial values of the parameters are set. Through the Lunar Lander and Car Racing tasks, we show that our approach is more sample-efficient than an unstructured baseline and also more robust to noisy environments. We also presented qualitatively how having a structure enables more robustness to imperfect expert demonstrations. Finally, we discussed the limitations of this work and how it can be extended in the near future.

Acknowledgments

This research has been partially supported by Microsoft Corporation as part of the Keio CMU partnership.

References

[Ainsworth *et al.*, 2022] Samuel K Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries. *arXiv preprint arXiv:2209.04836*, 2022.

[Ankile *et al.*, 2024] Lars Ankile, Anthony Simeonov, Idan Shenfeld, and Pulkit Agrawal. Juicer: Data-efficient imitation learning for robotic assembly. *arXiv preprint arXiv:2404.03729*, 2024.

[Bahnsen *et al.*, 2016] Alejandro Correa Bahnsen, Djamila Aouada, Aleksandar Stojanovic, and Björn Ottersten. Feature engineering strategies for credit card fraud detection. *Expert Systems with Applications*, 51:134–142, 2016.

[Brandfonbrener *et al.*, 2023] David Brandfonbrener, Stephen Tu, Avi Singh, Stefan Welker, Chad Boodoo, Nikolai Matni, and Jake Varley. Visual backtracking teleoperation: A data collection protocol for offline image-based reinforcement learning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11336–11342. IEEE, 2023.

[Bucker *et al.*, 2024] Arthur Bucker, Pablo Ortega-Kral, Jonathan Francis, and Jean Oh. Grounding robot policies with visuomotor language guidance. *arXiv preprint arXiv:2410.06473*, 2024.

[Cheng *et al.*, 2024] Peng Cheng, Xianyuan Zhan, Wenjia Zhang, Youfang Lin, Han Wang, Li Jiang, et al. Look beneath the surface: Exploiting fundamental symmetry for sample-efficient offline rl. *Advances in Neural Information Processing Systems*, 36, 2024.

[Chi *et al.*, 2023] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.

[Dauner *et al.*, 2023] Daniel Dauner, Marcel Hallgarten, Andreas Geiger, and Kashyap Chitta. Parting with misconceptions about learning-based vehicle motion planning. In *Conference on Robot Learning*, pages 1268–1281. PMLR, 2023.

[Deng *et al.*, 2020] Changyu Deng, Xunbi Ji, Colton Rainey, Jianyu Zhang, and Wei Lu. Integrating machine learning with human knowledge. *Iscience*, 23(11), 2020.

[Deshpande *et al.*, 2024] Abhay Deshpande, Liyiming Ke, Quinn Pfeifer, Abhishek Gupta, and Siddhartha S Srinivasa. Data efficient behavior cloning for fine manipulation via continuity-based corrective labels. *arXiv preprint arXiv:2405.19307*, 2024.

[Eisner *et al.*, 2024] Ben Eisner, Yi Yang, Todor Davchev, Mel Vecerik, Jonathan Scholz, and David Held. Deep se (3)-equivariant geometric reasoning for precise placement tasks. *arXiv preprint arXiv:2404.13478*, 2024.

[Fu *et al.*, 2024] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. In *Conference on Robot Learning (CoRL)*, 2024.

[Ghojogh *et al.*, 2019] Benyamin Ghojogh, Maria N Samad, Sayema Asif Mashhadi, Tania Kapoor, Wahab Ali, Fakhri Karray, and Mark Crowley. Feature selection and feature extraction in pattern analysis: A literature review. *arXiv preprint arXiv:1905.02845*, 2019.

[Greenblatt *et al.*, 2024] Ryan Greenblatt, Carson Denison, Benjamin Wright, Fabien Roger, Monte MacDiarmid, Sam Marks, Johannes Treutlein, Tim Belonax, Jack Chen, David Duvenaud, et al. Alignment faking in large language models. *arXiv preprint arXiv:2412.14093*, 2024.

[Ho *et al.*, 2022] Mark K Ho, David Abel, Carlos G Correa, Michael L Littman, Jonathan D Cohen, and Thomas L Griffiths. People construct simplified mental representations to plan. *Nature*, 606(7912):129–136, 2022.

[Hu *et al.*, 2022] Anthony Hu, Gianluca Corrado, Nicolas Griffiths, Zachary Murez, Corina Gurau, Hudson Yeo, Alex Kendall, Roberto Cipolla, and Jamie Shotton. Model-based imitation learning for urban driving. *Advances in Neural Information Processing Systems*, 35:20703–20716, 2022.

[Kolev *et al.*, 2024] Victor Kolev, Rafael Rafailov, Kyle Hatch, Jiajun Wu, and Chelsea Finn. Efficient imitation learning with conservative world models. *arXiv preprint arXiv:2405.13193*, 2024.

[Liu *et al.*, 2023] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

[Liu *et al.*, 2024] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

[Mao *et al.*, 2023] Jiayuan Mao, Tomás Lozano-Pérez, Joshua B Tenenbaum, and Leslie Pack Kaelbling. Pdsketch: Integrated planning domain programming and learning. *arXiv preprint arXiv:2303.05501*, 2023.

[OpenAI, 2024] OpenAI. Gpt-4o system card, 2024.

[Osa *et al.*, 2018] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, Jan Peters, et al. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179, 2018.

[Park and Wong, 2022] Jung Yeon Park and Lawson Wong. Robust imitation of a few demonstrations with a backwards model. *Advances in Neural Information Processing Systems*, 35:19759–19772, 2022.

[Peng *et al.*, 2024] Andi Peng, Ilia Sucholutsky, Belinda Z Li, Theodore R Sumers, Thomas L Griffiths, Jacob Andreas, and Julie A Shah. Learning with language-guided state abstractions. *arXiv preprint arXiv:2402.18759*, 2024.

[Schneuing *et al.*, 2024] Arne Schneuing, Charles Harris, Yuanqi Du, Kieran Didi, Arian Jamasb, Ilia Igashov, Weitao Du, Carla Gomes, Tom L Blundell, Pietro Lio, et al. Structure-based drug design with equivariant diffusion models. *Nature Computational Science*, 4(12):899–909, 2024.

[Tang *et al.*, 2024a] Hao Tang, Keya Hu, Jin Peng Zhou, Sicheng Zhong, Wei-Long Zheng, Xujie Si, and Kevin Ellis. Code repair with llms gives an exploration-exploitation tradeoff, 2024.

[Tang *et al.*, 2024b] Hao Tang, Darren Key, and Kevin Ellis. Worldcoder, a model-based llm agent: Building world models by writing code and interacting with the environment. *arXiv preprint arXiv:2402.12275*, 2024.

[Towers *et al.*, 2024] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2024.

[Wan *et al.*, 2024] Weikang Wan, Yifeng Zhu, Rutav Shah, and Yuke Zhu. Lotus: Continual imitation learning for robot manipulation through unsupervised skill discovery. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 537–544. IEEE, 2024.

[Wang *et al.*, 2023] Sifan Wang, Shyam Sankaran, Hanwen Wang, and Paris Perdikaris. An expert's guide to training physics-informed neural networks. *arXiv preprint arXiv:2308.08468*, 2023.

[Wang *et al.*, 2024] Yufei Wang, Zhanyi Sun, Jesse Zhang, Zhou Xian, Erdem Biyik, David Held, and Zackory Erickson. Rl-vlm-f: Reinforcement learning from vision language foundation model feedback. *arXiv preprint arXiv:2402.03681*, 2024.

[Wei *et al.*, 2023] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[Xie *et al.*, 2024] Annie Xie, Lisa Lee, Ted Xiao, and Chelsea Finn. Decomposing the generalization gap in imitation learning for visual robotic manipulation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3153–3160. IEEE, 2024.

[Zhao *et al.*, 2024] Tony Z Zhao, Jonathan Tompson, Danny Driess, Pete Florence, Kamyar Ghasemipour, Chelsea Finn, and Ayzaan Wahid. Aloha unleashed: A simple recipe for robot dexterity. *arXiv preprint arXiv:2410.13126*, 2024.

[Zhou *et al.*, 2023] Allan Zhou, Moo Jin Kim, Lirui Wang, Pete Florence, and Chelsea Finn. Nerf in the palm of your hand: Corrective augmentation for robotics via novel-view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17907–17917, 2023.

[Zhou *et al.*, 2024] Zihao Zhou, Bin Hu, Chenyang Zhao, Pu Zhang, and Bin Liu. Large language model as a policy teacher for training reinforcement learning agents, 2024.

[Zhu and Simmons, 2024] Feiyu Zhu and Reid Simmons. Bootstrapping cognitive agents with a large language model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 655–663, 2024.