Towards Using Probabilistic Models to Design Software Systems with Inherent Uncertainty

Alex Serban[1,2], Erik Poll[1], and Joost Visser[3]

[1] Radboud University,
[2] Software Improvement Group
[3] Leiden University
The Netherlands
a.serban@cs.ru.nl

Abstract. The adoption of machine learning (ML) components in software systems raises new engineering challenges. In particular, the inherent uncertainty regarding functional suitability and the operation environment makes architecture evaluation and trade-off analysis difficult. We propose a software architecture evaluation method called Modeling Uncertainty During Design (MUDD) that explicitly models the uncertainty associated to ML components and evaluates how it propagates through a system. The method supports reasoning over how architectural patterns can mitigate uncertainty and enables comparison of different architectures focused on the interplay between ML and classical software components. While our approach is domain-agnostic and suitable for any system where uncertainty plays a central role, we demonstrate our approach using as example a perception system for autonomous driving.

Keywords: Software architecture · Machine Learning · Uncertainty.

1 Introduction

With the emergent adoption of ML components in software systems, there is an increased need to tackle and harness their *inherent* uncertainty. Methods to address uncertainty exist for design time [7, 4] and for run-time [3]. However, previous work focused primarily on uncertainty related to the parameters used to model a system or its context [3, 7, 4]. ML components add a new type of uncertainty that was only briefly explored previously; stemming from the fundamental impossibility to fully verify that they can satisfy their intended functionality and that they are able to cope with stochastic events during operation [11].

In this paper we introduce a method to evaluate architecture design alternatives for software using both traditional and ML components. The proposal, called Modeling Uncertainty During Design (MUDD), is based on two guiding principles. Firstly, the threats due to inherent uncertainty of ML components are evaluated both locally (for the specific components) and tracked as they propagate and influence other components in the system. Secondly, the prior

information about uncertainty of ML components which is used at design time is considered incomplete and subject to continuous change.

The rest of the paper is organized as follows. Firstly, MUDD is introduced (Section 2), followed by a demonstration (Section 3), related work (Section 4) and conclusions (Section 5).

2 Modeling Uncertainty During Design (MUDD)

MUDD explicitly models two sources of uncertainty specific to "automated learning" [6]: (1) epistemic uncertainty, i.e., the uncertainty about the data generation process (used for training ML models), and (2) stochastic uncertainty, i.e., the uncertainty related to stochastic noise in the environment where a ML component operates. These uncertainty types have been studied in self-adaptive systems [8], where software architecture plays an important role. MUDD is distinct by modeling these uncertainties at *design* time, rather than at run time.

Notably, MUDD supports reasoning over which design alternatives are less sensitive to uncertainty and how design patterns can help mitigate it. Moreover, the method allows to evaluate hypothetical scenarios, in which the data about uncertainty used at design time is incomplete or assumed to take any value.

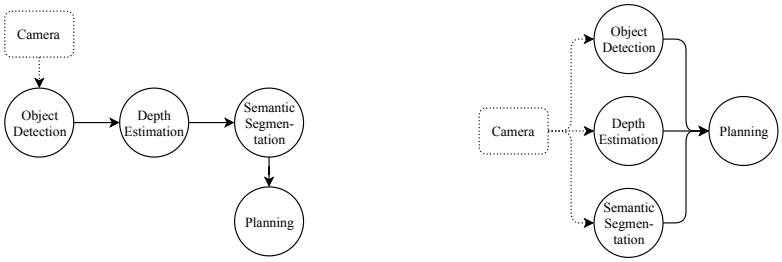
From a methodological perspective, MUDD only requires to annotate existing software architectures with the sources of uncertainty specific to ML components. Under the hood, MUDD uses Bayesian networks (BNs) to model a software system, propagate the uncertainties and obtain quantitative data about the architecture's sensitivity to uncertainty.

We emphasize that MUDD uses these two uncertainty types because they are application and context *independent*, i.e., they are valid for any ML model. The methods used to measure them can be different, depending on the ML algorithm employed. Therefore, they are parameters rather than fixed elements of MUDD. Nonetheless, MUDD is not limited to any type of uncertainty.

Throughout the paper we use an example from autonomous driving, inspired by [1, 10] – the design of a perception system for scene understanding. The system performs three tasks: (1) object detection, which aims to identify the location of all objects in an image, (2) semantic segmentation, which assigns each pixel in an image to a predefined class, and (3) depth estimation, which determines the position of obstacles or the road surface.

The outcome of the example perception system is used in planning the next driving maneuvers. The functionality of all components is implemented using deep learning (DL) because no specification can be written for it, and other ML algorithms perform worse. We are interested to evaluate software architecture design alternatives and select the one which is the least sensitive to uncertainty.

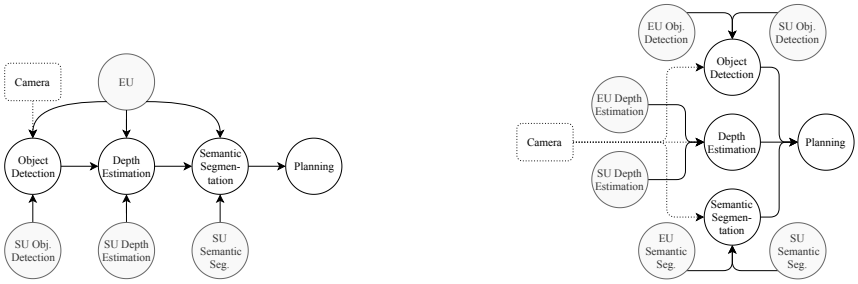
In Figure 1a and Figure 1b we present two architecture candidates inspired by [10] and [1]. The relevant functional components are illustrated using circles while the input coming from the camera is depicted with a rectangle. The latter will not be later considered a node in the BN (therefore its shape).



(a) End-to-end architecture.

(b) Component-based architecture.

Fig. 1: Functional architectures for a scene understanding system in autonomous vehicles.



(a) End-to-end annotated architecture, as required by MUDD.

(b) Component-based annotated architecture, as required by MUDD.

Fig. 2: Uncertainty representation for the two architectures presented in Figure 1, where EU stands for epistemic uncertainty and SU for stochastic uncertainty.

The first figure illustrates the end-to-end paradigm, where all components of the system are jointly trained to form a representation relevant to planning. This corresponds to the recommendation in [10]. The components share a base network for feature extraction and have independent layers to decode the features for each task. An alternative architecture is presented in the Figure 1b, where the system is organized into distinct ML components and integrated during planning. This corresponds to the architecture recommended in [1]. We have chosen these architectural styles as the only alternatives we could find in literature. However, MUDD is not limited to any architectural style.

For reasoning about uncertainties, we propose to annotate the the two architectures with the sources of uncertainty specific to each component. An example is given in Figure 2, which departs from the functional view presented in Figure 1 by illustrating the uncertainty sources, for each component. In the first case, Figure 2a, one base encoder is used for all tasks. Therefore, only one node representing epistemic uncertainty (EU) influences all components.

Different sources of stochastic uncertainty (SU) can impact the three tasks because one random event in the operational environment can influence segmentation, but not detection or depth estimation (and vice versa). Therefore, for each component there is a different variable for stochastic uncertainty. In the second scenario from Figure 2b, the components process raw data from camera independently. Therefore, they are subject to distinct epistemic and stochastic uncertainties. We note that these decisions are not application and context specific. All ML components are subject to these types of uncertainty.

3 Quantitative Architecture Evaluation

Under the hood MUDD uses BNs to process quantitative data about uncertainties. The probabilities needed to populate the network can be defined by experts or inferred through simulations. The random variables in the BN can take continuous or discrete values. In the former case, the system designer chooses an a priori distribution for each variable, before seeing any data, and updates its parameters once new observations are available. In the latter case, the variables take discrete values and are described by their probability mass functions.

For simplicity, we choose to model all variables through probability mass functions with two discrete values: *low* or *high* uncertainty. When the uncertainty is low, the system is likely to satisfy its intended functionality and vice versa. Given the two proposed values for uncertainty, we are interested in evaluating the influence of different nodes in the network on planning and obtain quantitative results for the qualitative evaluation presented earlier. Both the probabilities and the thresholds can be decided by domain experts or by simulation.

For the running example we use a test data set to extract the uncertainty estimates from DL components, by averaging over samples in this data set. The thresholds between low and high represent the lowest uncertainty estimate from the incorrectly classified examples in the testing data set. The probability that a component has high (epistemic or stochastic) uncertainty will be the total number of test examples which have uncertainty higher than the threshold over the total number of testing examples. Note that the correctly classified examples with high uncertainty will contribute to the probability that a component has high uncertainty. This choice is deliberate because the system we study is safety-critical and uncertain decisions should be avoided altogether.

The conditional probabilities – i.e., the influence of components to the connected components – are evaluated in a similar manner. They represent the probabilities that a component has high uncertainty, given the uncertainty values of the parent variables. For example, $P(OD = \mathrm{H} | EP = \mathrm{H}, SU = \mathrm{H})$ is the probability that the object detector is highly uncertain when the model has high epistemic and high stochastic uncertainty. We use the same method and data set as before, but average the results when the parent variables have the same value. The thresholds are also chosen as before.

Uncertainty estimation. All experiments are carried out using the CityScape data set [2]. For the end-to-end architecture presented in Figure 1a we train a

variant of MultiNet [12] using an encoder based on the DenseNet architecture, pre-trained on the ImageNet data set with a dropout probability of $p = 0.2$. We use different loss functions in a multi-task learning setting for object detection, depth estimation and semantic segmentation. Epistemic uncertainty is approximated by casting a Bernoulli distribution over the model's weights and sample it at evaluation time using the dropout layers in the base encoder. The mean of the dropout samples is used for prediction and the variance to output the uncertainty for each class. Stochastic uncertainty is extracted from the final layer of each task. For the component-based architecture presented in Figure 1b we use an independent encoder and decoder for each task. Training is performed by minimizing the task specific loss function used in the multi-task setting described above. The implementation of DL components was done in Pytorch[4] and the BNs in Pomegranate[5]. The uncertainty estimates are presented in Table 1 for the system in Figure 1a and Table 2 for the system in Figure 1b.

The heuristics applied to populate the tables represent the prior knowledge we embed in the network. Depending on the context, software designers may choose to embed more domain knowledge or rely on expert opinion.

Given the probability tables, we can use the inference rules of BNs to answer questions about the proposed architectures. We provide a working example: e.g., we wish to get quantitative evidence about the impact of high stochastic uncertainty in depth estimation on planning. Setting depth estimation stochastic uncertainty to "High" ($SU_{DE} = \mathrm{H}$), we can compute the final impact on planning as follows. Let $\pi(x)$ represent the parent variables of node x (the nodes that have a directed edge to it). The probability that planning will have high uncertainty is:

$$P(Planning = \mathrm{H}) = P(SS|\ \pi(SS)) \cdot P(DE|\ \pi(DE)) \cdot P(OD|\ \pi(OD)) \cdot P(SU_{SS}) \cdot P(SU_{DE} = \mathrm{H}) \cdot P(SU_{OD}) \cdot P(EU),$$

for the end-to-end architecture and:

$$P(Planning = \mathrm{H}) = P(SS|\ \pi(SS)) \cdot P(DE|\ \pi(DE)) \cdot P(OD|\ \pi(OD)) \cdot P(SU_{SS}) \cdot P(SU_{DE} = \mathrm{H}) \cdot P(SU_{OD}) \cdot P(EU_{SS}) \cdot P(EU_{DE}) \cdot P(EU_{OD}),$$

for the component-based architecture, where the acronyms are as in Table 1 or 2.

Running the computation we observe that the probability of uncertain planning is approximately 10% lower for the component-based architecture (Figure 1b) than for the end-to-end architecture. Moreover, through the same model we can analyze how high stochastic uncertainty in depth estimation impacts planning within the minimum and maximum bounds. We plot the probability that planning is uncertain given that depth estimation stochastic uncertainty is high, by varying $P(DE = \mathrm{H}|SU = \mathrm{H},\ \cdot\ )$ in Tables 1 and 2 between $[0, 1]$ with a step size of 0.01. The results are illustrated in Figure 3a.

The plot represents the influence of high stochastic uncertainty on depth estimation and the way it propagates on planning. We observe that in the

[4] https://pytorch.org/
[5] https://github.com/jmschrei/pomegranate

$P(\cdot)$	EU	SU_{OD}	SU_{DE}	SU_{SS}
H	0.18	0.16	0.11	0.19

$P(Planning \mid SS)$	
0.1	L
0.9	H

$P(OD \mid EU, SU_{OD})$		
0.0	L	L
0.64	L	H
0.61	H	L
1	H	H

$P(DE \mid EU, SU_{DE}, OD)$			
0.0	L	L	L
0.13	L	L	H
0.76	L	H	L
0.85	L	H	H
0.43	H	L	L
0.78	H	L	H
0.9	H	H	L
1	H	H	H

$P(SS \mid EU, SU_{SS}, DE)$			
0.0	L	L	L
0.28	L	L	H
0.64	L	H	L
0.72	L	H	H
0.66	H	L	L
0.58	H	L	H
0.61	H	H	L
1	H	H	H

Table 1: Independent and conditional probabilities for the end-to-end architecture in Figure 2a. The acronyms used are OD – object detection, DE – depth estimation, SS – semantic segmentation, EU – epistemic uncertainty and SU – stochastic uncertainty. The uncertainty values are L - low and H - high.

$P(\cdot)$	EU_{OD}	SU_{OD}	EU_{DE}	SU_{DE}	EU_{SS}	SU_{SS}
H	0.14	0.16	0.31	0.44	0.17	0.19

$P(OD \mid EU_{OD}, SU_{OD})$		
0.0	L	L
0.57	L	H
0.41	H	L
1.0	H	H

$P(DE \mid EU_{DE}, SU_{DE})$		
0.0	L	L
0.51	L	H
0.47	H	L
1	H	H

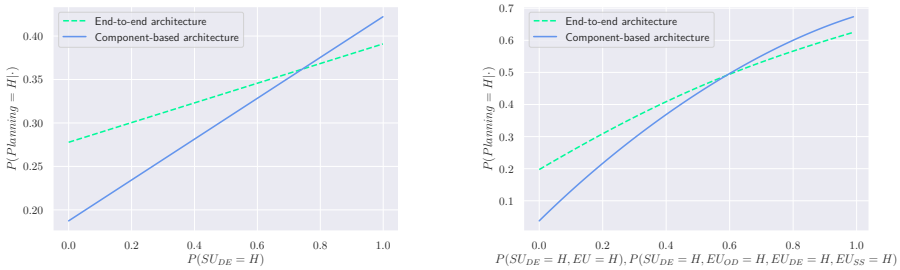
$P(SS \mid EU_{SS}, SU_{SS})$		
0.0	L	L
0.11	L	H
0.42	H	L
1.0	H	H

$P(Planning \mid SS, DE, OD)$			
0.0	L	L	L
0.34	L	L	H
0.34	L	H	L
0.66	L	H	H
0.34	H	L	L
0.66	H	L	H
0.66	H	H	L
1	H	H	H

Table 2: Independent and conditional probabilities for the component-based architecture in Figure 2b. The acronyms used are described in Table's 1 caption.

component-based architecture stochastic uncertainty in depth estimation has a lower impact on planning than in the end-to-end architecture, for values up to ~ 0.7, after which the end-to-end architecture is more resilient to uncertainty. Depending on the operational environment, a software architect can choose the design that better fits the expected conditions. For example, if an autonomous vehicle operates in limited domains – e.g., inside a warehouse – where the probability of encountering stochastic events is low, the component-based architecture for the scene understanding system is more appropriate.

Using the same model we can evaluate the influence of multiple sources of uncertainty on planning. We use the realistic assumption that the CityScape data set does not approximate all driving scenarios and thus may introduce high



(a) Influence of high stochastic uncertainty in depth estimation on planning.

(b) Influence of high stochastic uncertainty for depth estimation and all epistemic uncertainties on planning.

Fig. 3: Quantitative evaluation of uncertainty in software architecture design.

epistemic uncertainties. Therefore, we evaluate the influence of all epistemic uncertainty sources on planning in the scenario described above, where we assume high stochastic uncertainty in depth estimation. We use the same method as above to evaluate the probability that planning will have high uncertainty while we vary all epistemic uncertainty nodes simultaneously with the stochastic uncertainty in depth estimation. The uncertainties vary between $[0, 1]$, with a step size of 0.01. The results are plotted in Figure 3b.

As in the previous case, the end-to-end architecture is more resilient to high uncertainties, for all the components mentioned above. Moreover, the threshold where the end-to-end architecture becomes more resilient than the component-based architecture is lower. However, epistemic uncertainty can be removed using more training data, so the scenario in which epistemic uncertainty is low is more realistic. In this case, the component-based architecture is more resilient to uncertainty than the end-to-end architecture.

4 Related Work

At design time, the uncertainty in the parameters used to model a software system has been taken into account for evaluating the reliability of software architectures using robust optimization [7], for comparing software architectures when the impact of architectural decisions can not be quantified, using fuzzy methods [4] and for evaluating trade-offs specific to quality attributes such as performance, using sensitivity analysis [5]. However, none of these methods take into account the uncertainty related to "automated learning", as indicated by [6].

At run-time, various sources of uncertainty can be mitigated through self-adaptation [3]. While several methods for self-adaptation use a related formalism, we tackle the problem at design time, and *not* at run-time, as in self adaptation. Therefore, self-adaptation is complementary, and a method that can unify uncertainty at design and run time is an interesting direction for future research.

5 Conclusions and Future Work

We introduce MUDD, a method to evaluate and compare architecture design alternatives for systems using ML components. In particular, we propose to explicitly model the inherent uncertainty specific to ML components at design time, and evaluate how it propagates and influences other components in a system. The proposed information needed to quantify the uncertainty for each ML component is well studied both in the software architecture and in the ML literature. For modeling software systems, MUDD uses Bayesian networks (BNs).

For future work we propose to further validate the sources of uncertainty with practitioners (e.g., through interviews), and to facilitate the use of MUDD by developing or integrating with appropriate tools. New scenarios, which can better exhibit the potential of MUDD and new uncertainty sources (e.g., [9]) are planned as well. Also, BNs are directed graphs and do not allow loops. Alternatives that can overcome this limitation are planned for future work.

References

1. Behere, S., Törngren, M.: A functional reference architecture for autonomous driving. Information and Software Technology **73**, 136–150 (2016)
2. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The cityscapes dataset for semantic urban scene understanding. In: IEEE CVPR. pp. 3213–3223 (2016)
3. Esfahani, N., Malek, S.: Uncertainty in self-adaptive software systems. In: Software Engineering for Self-Adaptive Systems II, pp. 214–238. Springer (2013)
4. Esfahani, N., Malek, S., Razavi, K.: Guidearch: guiding the exploration of architectural solution space under uncertainty. In: ICSE. pp. 43–52. IEEE (2013)
5. Etxeberria, L., Trubiani, C., Cortellessa, V., Sagardui, G.: Performance-based selection of software and hardware features under parameter uncertainty. In: International Conference on Quality of Software Architectures. pp. 23–32 (2014)
6. Mahdavi-Hezavehi, S., Avgeriou, P., Weyns, D.: A classification framework of uncertainty in architecture-based self-adaptive systems with multiple quality requirements. In: Managing Trade-Offs in Adaptable Software Architectures, pp. 45–77. Elsevier (2017)
7. Meedeniya, I., Aleti, A., Grunske, L.: Architecture-driven reliability optimization with uncertain model parameters. JSS **85**(10), 2340–2355 (2012)
8. Perez-Palacin, D., Mirandola, R.: Uncertainties in the modeling of self-adaptive systems: A taxonomy and an example of availability evaluation. In: ACM/SPEC International Conference on Performance Engineering. pp. 3–14 (2014)
9. Serban, A., Poll, E., Visser, J.: Adversarial examples on object recognition: A comprehensive survey. ACM Computing Surveys (CSUR) **53** (2020)
10. Serban, A., Poll, E., Visser, J.: A standard driven software architecture for fully autonomous vehicles. In: JASE. pp. 20–33. Atlantis Pres (2020)
11. Serban, A.C.: Designing safety critical software systems to manage inherent uncertainty. In: IEEE ICSA-C. pp. 246–249. IEEE (2019)
12. Teichmann, M., Weber, M., Zoellner, M., Cipolla, R., Urtasun, R.: Multinet: Real-time joint semantic reasoning for autonomous driving. In: Intelligent Vehicles. pp. 1013–1020. IEEE (2018)